UNIVERSIDAD AUTÓNOMA DE SINALOA

FACULTAD DE INFORMÁTICA CULIACÁN
FACULTAD DE CIENCIAS DE LA TIERRA Y EL ESPACIO

MAESTRÍA EN CIENCIAS DE LA INFORMACIÓN

IMPLEMENTACIÓN DE UN NÚCLEO DE OPERADORES FUNDAMENTALES PARA REALIZAR BÚSQUEDA POR SEMEJANZA SOBRE SEÑALES DE ELECTROCARDIOGRAMAS DIGITALES DENTRO DE UN SISTEMA GESTOR DE BASES DE DATOS

Que como requisito parcial para obtener el grado de
MAESTRO EN CIENCIAS DE LA INFORMACIÓN
presenta

LUIS ENRIQUE IBARRA CÁZARES

DIRECTOR DE TESIS: INÉS FERNANDO VEGA LÓPEZ

Culiacán, Sinaloa a Julio de 2017

Agradecimientos

Quiero primeramente agradecer a mi familia, a quieres cada día me ofrecieron su apoyo incondicional, y quienes cada día al llegar a casa me preguntaron ¿cómo te fue hoy hijo?. A mi abuela Maria Luisa, a mi madre Lourdes, a mi hermana Cinthia y a todas mis tías. Especial agradecimiento tengo también a mi esposa Lizbeth, quien ha estado conmigo incondicionalmente durante todo este tiempo, apoyándome, dando ánimos y confiando en mí para poder superarme y ser quien soy el día de hoy.

De igual manera, agradezco a mi asesor de tesis, el Dr. Inés Fernando Vega López, por su apoyo, tiempo, paciencia y consejos que me han llevado a concluír este trabajo y otros distintos proyectos. Una persona respetable y admirable a quien aprecio mucho. Al Dr. Jorge Adalberto Navarro Castillo, al M.C. Gerardo Beltrán Gutiérrez y al Dr. Manuel E. Trejo Soto, de quienes obtuve los mejores aprendizajes durante el tiempo de estudios de mi maestría.

A mis compañeros de posgrado Álvaro, Juan Carlos, Vilma, Daniel, Selene, José Carlos, Gabriela, Silvestre, Paúl, Saúl, Ernesto , Eutimio. Junto a ellos aprendí, avancé y logré estar donde estoy el día de hoy, con su amistad, con sus consejos, palabras de aliento y motivación.

Finalmente, quiero agradecer al Consejo Nacional de Ciencia y Tecnología por su apoyo financiero en la beca 296007. Además, agradezco a la Universidad Autónoma de Sinaloa, así como a la Facultad de Informática Culiacán y a la Facultad de Ciencias de la Tierra y el Espacio, por brindarme sus instalaciones y apoyo durante el desarrollo de este trabajo.

Lista de Acrónimos

ECG: Electrocardiograma.

SGBD: Sistema Gestor de Bases de Datos.

SGBDR: Sistema Gestor de Bases de Datos Relacional.

SQL: Structured Query Language.

UDF: User Defined Function.

MBR: Minimum Bounding Rectangle.

BLOB: Binary Large Object.

OLAP: On-line Analytical Processing

OLTP: Online Transaction Processing.

KNN: K Nearest Neighbors.

DWT: Dynamic Time Warping.

ICC: Índice de Correlación Cruzada.

CTE: Common Table Expression.

Índice general

Capítulo 1. Introducción **1**
1.1. Definición Formal del Problema 3
1.2. Motivación 4
1.3. Principales Logros y Aportaciones del Presente Trabajo 6
1.4. Organización Restante del Trabajo 7

Capítulo 2. Estado del Arte **9**
2.1. Procesamiento de Señales de ECG Digitales 10
2.1.1. Etapas Previas a la Búsqueda por Semejanza en el Procesamiento de Señales de ECG Digitales 11
2.1.2. Clasificación y Búsqueda por Semejanza 14
2.2. Sistemas Gestores de Bases de Datos 19
2.2.1. Sistemas Gestores de Bases de Datos Row Store y Column Store 23
2.2.2. Diferencias Entre Row Store y Column Store 24
2.2.3. Sistemas Gestores de Bases de Datos Híbridos 25
2.3. Áreas de Oportunidad 27

Capítulo 3. Implementación de Operaciones para Realizar Búsqueda por Semejanza Sobre Señales de ECG Digitales **28**
3.1. Propuesta de Modelos de Almacenamiento 29
3.1.1. Almacenamiento Horizontal 30
3.1.2. Almacenamiento Vertical 31
3.2. Desarrollo de Algoritmos de Búsqueda por Semejanza 31
3.2.1. Medidas de Semejanza 33
3.2.2. Búsqueda por Vecinos más Cercanos (K-NN) 39
3.2.3. Búsqueda por Rango (ϵ-search) 41
3.2.4. Operadores Básicos en Algoritmos de Búsqueda por Semejanza 42
3.3. Extensión de un SGBD Utilizando Funciones Definidas por el Usuario 46
3.4. Propuesta General de Consultas Necesarias 49
3.4.1. K-NN Sobre el Modelo de Almacenamiento Horizontal 49
3.4.2. K-NN Sobre el Modelo de Almacenamiento Vertical 52
3.4.3. ϵ-search Sobre el Modelo de Almacenamiento Horizontal . . . 53
3.4.4. ϵ-search Sobre el Modelo de Almacenamiento Vertical 55

Capítulo 4. Evaluación Experimental y Resultados **58**
4.1. Descripción del Conjunto de Datos de Prueba 60
4.2. Modelos de Almacenamiento en el Sistema Gestor de Bases de Datos 64
4.2.1. Implementación del Modelo de Almacenamiento Horizontal . . 65

4.2.2. Implementación del Modelo de Almacenamiento Vertical . . . 66
4.2.3. Selección de Patrones Para los Algoritmos de Búsqueda por Semejanza . . . 67
4.2.4. Requerimientos Físicos de Almacenamiento . . . 68
4.3. Búsqueda de Subsecuencias con Consultas SQL . . . 70
4.3.1. Alternativas de Manejo de Datos Intermedios entre Subconsultas 71
4.3.2. Consultas Sobre el Modelo de Almacenamiento Horizontal Para el Algoritmo K-NN . . . 74
4.3.3. Consultas Sobre el Modelo de Almacenamiento Vertical Para el Algoritmo K-NN . . . 86
4.3.4. Consultas Sobre el Modelo de Almacenamiento Horizontal Para el Algoritmo ϵ-Search . . . 89
4.3.5. Consultas Sobre el Modelo de Almacenamiento Vertical Para el Algoritmo ϵ-search . . . 100
4.3.6. Ejecución de Consultas Sobre el SGBD . . . 102
4.3.7. Tiempos de Ejecución Para el Algoritmo K-NN . . . 104
4.3.8. Tiempos de Ejecución Para el Algoritmo ϵ-search . . . 109
4.3.9. Tiempos de Ejecución Sobre una Base de Datos de Gran Tamaño 114

Capítulo 5. Conclusiones . . . 119

Índice de figuras

Figura 1.1. Representación de un Ciclo Cardíaco con las Ondas P-QRS-T. . 5

Figura 2.1. Etapas del Procesamiento de Señales de ECG Digitales 11
Figura 2.2. Representación Gráfica de una Relación llamada "Personas". . 21
Figura 2.3. Almacenamiento en un *SGBD Row-Store*. 23
Figura 2.4. Almacenamiento en un *SGBD Column-Store*. 24

Figura 3.1. Pasos Necesarios para Realizar Algoritmos de Búsqueda por Semejanza Sobre Señales de Electrocardiogramas Digitales Dentro de un Sistema Gestor de Bases de Datos. 30
Figura 3.2. División de Áreas en el Campo de Búsqueda por Semejanza. . . 32
Figura 3.3. Tres Secuencias Denominadas S_1,S_2 y S_3 de Misma Longitud $n = 100$. 35
Figura 3.4. Representación Gráfica del Cálculo de Semejanza Punto a Punto. 36
Figura 3.5. Tres Secuencias Denominadas S_1,S_2 y S_3 con Longitudes $n_1 = 70$,$n_2 = 100$ y $n_3 = 90$. 37
Figura 3.6. Representación Gráfica del Cálculo de Semejanza Utilizando DTW. 38
Figura 3.7. Aglomeración de Secuencias De Acuerdo a los Distintos Algoritmos de Cálculo de Semejanza. 39
Figura 3.8. Operadores Identificados en el Cálculo de K-NN Sobre el Modelo de Almacenamiento Horizontal. 43
Figura 3.9. Operadores Identificados en el Cálculo de K-NN Sobre el Modelo de Almacenamiento Vertical. 44
Figura 3.10. Operadores Identificados en el Cálculo de ϵ-search Sobre el Modelo de Almacenamiento Horizontal. 45
Figura 3.11. Operadores Identificados en el Cálculo de ϵ-search Sobre el Modelo de Almacenamiento Vertical. 46

Figura 4.1. Subsecuencia de la Señal de ECG Digital de la Base de Datos ANSI/AAMI EC13 Identificada como 'aami3a'. 61
Figura 4.2. Subsecuencia de la Señal de ECG Digital de la Base de Datos QT Identificada como 'sel100'. 62
Figura 4.3. Subsecuencia de la Señal de ECG Digital de la Base de Datos MIT-BIH Arrhythmia Identificada como '100'. 63
Figura 4.4. Subsecuencia de la Señal de ECG Digital de la Base de Datos European ST-T Identificada como 'e0103'. 64

Índice de Tablas

Tabla 3.1. Reutilización de Operadores Identificados Entre Algoritmos y Modelos de Almacenamiento 57
Tabla 4.1. Procedencia de las 105 Señales que Componen la Base de Datos QT. 62
Tabla 4.2. Requerimientos Físicos de Almacenamiento de Datos en Disco (en Megabytes). 69
Tabla 4.3. Resumen General de las Consultas Propuestas. 74
Tabla 4.4. Tiempos de Ejecución en Segundos Para el Algoritmo 1-NN . . . 105
Tabla 4.5. Tiempos de Ejecución en Segundos para el Algoritmo 10-NN . . 106
Tabla 4.6. Tiempos de Ejecución en Segundos para el Algoritmo 100-NN . . 108
Tabla 4.7. Tiempos de Ejecución en Segundos para el Algoritmo E1-Search 110
Tabla 4.8. Tiempos de Ejecución en Segundos para el Algoritmo E2-Search 112
Tabla 4.9. Tiempos de Ejecución en Segundos para el Algoritmo E3-Search 113
Tabla 4.10. Tiempos de Ejecución en Segundos para el Algoritmo K-NN y el Modelo de Almacenamiento Horizontal en una Base de Datos de Gran Tamaño 116
Tabla 4.11. Proporciones de Incremento en Tiempos de Ejecución en la Base de Datos GranDB Respecto a los Tiempos de Ejecución Sobre la Base de Datos European 117

Capítulo 1
Introducción

En las últimas décadas se ha observado un crecimiento significativo en el desarrollo de métodos eficientes para el procesamiento y análisis de señales de electrocardiogramas (ECG) digitales, ya que la información que ofrecen sobre la condición médica del corazón ayuda a la prevención o detección de condiciones cardíacas [52]. Una señal de ECG se obtiene mediante la colocación de electrodos en el pecho, brazos y piernas, generando así diversas derivaciones. Una derivación es el registro de la diferencia de potenciales eléctricos entre dos puntos, ya sea entre dos electrodos o entre un punto virtual y un electrodo [70]. Cabe aclarar que a lo largo de este trabajo, nos referimos como señal de ECG digital a cada derivación de manera independiente.

La disponibilidad de bases de datos de señales es de gran importancia tanto para el desarrollo como para la evaluación de algoritmos del procesamiento de señales [56]. Algunas de estas bases de datos pueden encontrarse en la web gracias a repositorios como *Physionet*, que es un portal en línea creado para la concentración e intercambio de señales biomédicas en un banco de datos denominado *PhysioBank* [21, 42, 43]. En *Physionet* se pueden encontrar diversas bases de datos con diversos tipos de señales biomédicas, entre ellas, señales de ECG digitales. Sin embargo, estas señales se encuentran en un formato no estructurado y no estandarizado, por lo que requieren un tratamiento especial y una conversión para que sea posible su utilización en casos específicos.

La administración y estandarización de datos se facilita gracias al uso de Sistemas Gestores de Bases de Datos (SGBD). Un SGBD se define como un software especializado en la creación, administración y consulta de grandes colecciones de datos de manera eficiente [19, 53]. Cuando se habla de utilizar un SGBD para el almacena-

miento de señales de ECG digitales, debe establecerse un modelo de almacenamiento abstracto que se encuentre propiamente estructurado y que permita la creación de herramientas para su posterior consulta.

Dentro del campo del procesamiento de señales de ECG digitales, la búsqueda por semejanza ha sido un foco de atención por parte de la comunidad médica, con el fin de buscar y encontrar patrones que puedan representar un potencial peligro en la salud cardiaca de los pacientes [18]. De manera concreta, la búsqueda por semejanza sobre señales de ECG digitales se encarga de buscar patrones definidos dentro de estas, así como la posición dentro de la señal donde el patrón fue encontrado. Es por ello que, guiados por la hipótesis de que es posible implementar operadores básicos que conformen los algoritmos para realizar búsqueda por semejanza sobre señales de ECG digitales como funciones propias de un SGBD, se proponen dos modelos de almacenamiento que preserven la estructura de cada señal de ECG. En este trabajo de tesis, nos referimos a estructura como las características principales de una señal de ECG digital. Dichas características son: a) Cada señal de ECG digital cuenta con un identificador único, b) Los valores dentro de cada señal de ECG digital se encuentran representados por un tipo de dato nativo del SGBD, c) Los valores dentro de cada señal de ECG cuentan con un índice que permite asignarles un orden secuencial. Con ello, se busca que los datos almacenados puedan ser replicables, comprensibles y a su vez, accedidos y representados por las herramientas y tipos de datos nativos proveídos por el SGBD. Junto a los modelos de almacenamiento, también se proponen distintas maneras de consultar la información con el fin de encontrar aquella combinación que nos ofrezca el mejor rendimiento en términos de eficiencia.

1.1. Definición Formal del Problema

Una serie de tiempo, puede representarse formalmente como una secuencia de valores reales [13]. Una secuencia s puede ser descrita formalmente como $s = (s[0], s[1], ..., s[n-1])$, donde $s[x]$ hace referencia al valor de s en la posición x y $s[x] \in \mathbb{R}$. Una subsecuencia propia de s se describe formalmente como $s' = s[i : j]$ tal que $0 \leq i \leq n-1, 0 \leq j \leq n-1, i \leq j$. Con base en lo anterior, podemos formalizar el problema atacado en este trabajo de la siguiente manera.

Sea S un conjunto con M señales de ECG digitales, cada una de las señales s_i puede ser tratada como una secuencia de valores, representada como $s_i = (s_i[0], s_i[1], ..., s_i[n_i - 1])$ tal que $s_i[x] \in \mathbb{R}, s_i \in S$. Sea O un conjunto de operadores utilizados dentro del campo de búsqueda por semejanza de la forma $O = \{o_j | 1 < j < N\}$. Cada uno de los operadores o_j puede ser considerado una función que puede tomar una de cuatro distintas formas. La primera forma $f_{o1}(t_1) = p_1$, cuyo dominio $t_1 \in T1 : S \times S$, y cuya imagen es una secuencia p_1 no necesariamente $p_1 \in S$. La segunda forma $f_{o2}(t_2) = p_2$, cuyo dominio $t_2 \in T2 : S \times \mathbb{N}$ y su imagen es una secuencia p_2 no necesariamente $p_2 \in S$. La tercera forma $f_{o3}(t_3) = p_3$, cuyo dominio $t_2 \in T_2 : S \times \mathbb{R}$, y su imagen es un valor numérico $p_3 \in \mathbb{N}$. Por último, la cuarta forma es $f_{o4}(t_4) = p4$, con dominio $t_4 \in T_4 : S \times \mathbb{N}$ e imagen $p_4 \in \mathbb{R}$

Con base en lo anterior, dado un conjunto de señales de ECG, S, y un conjunto de operadores O, se busca encontrar una forma de almacenar S dentro de un SGBD que permita definir operaciones para realizar búsqueda por semejanza y gestionar los datos de manera eficiente. Se busca también implementar los operadores o_j como herramientas propias del SGBD. De tal manera que cualquier operación dentro del campo de búsqueda por semejanza sobre señales de ECG digitales sea representada por una función global $f_g(t) = o'_N(o'_{N-1}(o'_{N-2}(...(o'_1(t, r'), r')...), r'), r')$, tal que $r' \in S \cup \mathbb{N} \cup \mathbb{R} \cup \{\}$, el dominio de $f_g = t \in T : S \times S \cup \{\} \times \mathbb{R} \cup \{\} \times \mathbb{N} \cup \{\}$, y su imagen se encuentre conformada por la composición de las imágenes de los N operadores

necesarios por la aplicación de la función sobre t y el parámetro acumulado r' [65].

1.2. Motivación

El electrocardiograma (ECG) es quizá la fuente de información biomédica más estudiada en el campo de la medicina, siendo un foco de atención desde hace ya más de 50 años [51]. Una señal de ECG digital puede tener distintas derivaciones [18, 55, 56]. El ECG refleja la actividad eléctrica del corazón y se caracteriza por ciclos cardíacos. Cada ciclo cardíaco en una señal de ECG consiste en un conjunto de ondas secuenciales identificadas como P-QRS-T. En la Figura 1.1 se ilustra la representación gráfica de un ciclo cardiaco con las ondas P-QRS-T, adaptado de [70]. La naturaleza misma del ECG puede evidenciar la presencia de anormalidades cardíacas, que pueden llegar a aparecer de manera aleatoria sobre una escala de tiempo [3]. Por lo tanto, el muestreo de datos del ECG y la variabilidad en el comportamiento del corazón podría llegar a ser analizado por largos periodos de tiempo. Por ejemplo, una señal de ECG digital con 6 derivaciones, cada una con una duración de 24 horas y muestreada a una frecuencia de 400Hz, pudiera requerir hasta 1,384 MB (Megabytes) de almacenamiento si se utiliza un tipo de dato numérico de doble precisión (8 bytes) para cada uno de los valores dentro de las derivaciones. Por esta razón, el volumen de los datos a manejar puede llegar a ser en la mayoría de los casos, muy alto. Como consecuencia, el estudio de estos volúmenes de datos puede resultar tedioso y consumidor de tiempo para un analista humano.

Si bien en los tiempos actuales el problema de la gran acumulación de señales de ECG digitales en bases de datos puede considerarse resuelto gracias al constante crecimiento en las capacidades de almacenamiento en sistemas computacionales actuales, debemos prestar especial atención en qué hacer con ellos una vez almacenados. Dada esta situación, el acceso eficiente a los datos, así como el desarrollo de herramientas para la extracción de información oportuna que permita la toma de decisiones

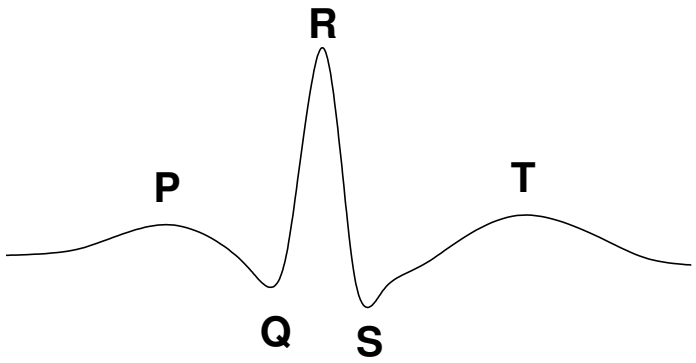


Figura 1.1: Representación de un Ciclo Cardíaco con las Ondas P-QRS-T.

son requeridos para proveer evidencia objetiva y apoyo fidedigno sobre las decisiones tomadas por médicos, investigadores, estudiantes de medicina y/o pacientes [13].

A pesar de la facilidad con la que pueden conseguirse actualmente los medios necesarios para el almacenamiento masivo de señales de ECG digitales, su manejo no se encuentra disponible y/o respaldado por los SGBD actuales. Estos sistemas no brindan un soporte nativo para el almacenamiento de señales de ECG digitales como tipo de dato estándar. Por consecuencia, los SGBD actuales tampoco brindan funciones para la consulta y tratamiento de señales de ECG digitales [39]. El modo común para el tratamiento de señales de ECG digitales se basa en el uso de herramientas ad-hoc, cada una con una función específica. Esto se ve afectado a la hora de exportar e importar información por un problema conocido como cuello de botella [12, 39, 47, 50].

Por las razones mencionadas anteriormente, los expertos en el área de la medicina no contemplan el uso de una herramienta única para el tratamiento de señales de ECG digitales. Para que se pueda contemplar una única herramienta como entorno de análisis y tratamiento, esta requiere contar con todas las capacidades y funciones requeridas durante el procesamiento. Por ello hemos propuesto agrupaciones de operadores básicos para no implementar cada uno de los algoritmos presentes en el tratamiento de señales de ECG digitales como una función ad-hoc sobre el SGBD. En

su lugar, se propone dar al SGBD la capacidad de proveer dichos algoritmos como una serie de operadores que pueden a su vez ser reutilizados en la construcción de diversas consultas que representan operaciones utilizadas dentro del campo del procesamiento de señales de ECG digitales.

1.3. Principales Logros y Aportaciones del Presente Trabajo

El presente trabajo propone la implementación de un conjunto de operadores básicos para realizar búsqueda por semejanza sobre señales de electrocardiogramas (ECG) digitales dentro de un Sistema Gestor de Bases de Datos (SGBD). Con el uso del SGBD podemos evitar problemas de pérdida de consistencia e integridad referencial. Los operadores desarrollados permiten estructurar el procesamiento de las señales de ECG digitales y nos permite a su vez plantear los algoritmos de análisis como consultas utilizando el lenguaje de consultas estándar SQL (Structured Query Language). A la vez, dichos operadores pueden ser reutilizados en distintos algoritmos de análisis que forman parte del campo del procesamiento de señales de ECG digitales.

Ante la falta de propuestas de estandarización para el tratamiento y almacenamiento de señales de ECG digitales dentro de una misma herramienta informática, este trabajo pretende aportar tal estandarización. La utilización de herramientas no integradas durante el análisis de señales de ECG digitales puede llevar a procesos complejos y tiempos muertos entre exportación e importación de datos [48, 50]. En este trabajo se ha logrado implementar la gestión y almacenamiento de señales de ECG digitales así como la realización de búsqueda por semejanza dentro de una sola herramienta, un SGBD. Se han podido integrar distintas operaciones que los SGBD nos ofrecen de manera nativa pero que son necesarias dentro del campo del procesamiento de señales de ECG digitales.

Proponemos también el uso de dos modelos de almacenamiento lógico de datos para la estructuración y correcta representación de las señales de ECG digitales dentro

del SGBD. Denominamos a estos modelos como modelo de almacenamiento horizontal y modelo de almacenamiento vertical, respectivamente. Con la implementación de ambos modelos de almacenamiento podemos lograr que las señales de ECG digitales puedan ser almacenadas y consultadas utilizando el lenguaje SQL.

Para ambos modelos de almacenamiento proponemos consultas estructuradas, expresadas en el lenguaje de consulta estándar SQL. Cada una de las consultas se componen por funciones nativas del SGBD y funciones definidas por el usuario (UDFs). Las consultas propuestas utilizan también distintos medios de almacenamiento temporal para su estructuración, ofrecidas por el propio SGBD. Con estas bases, se ha logrado realizar búsqueda por semejanza sobre señales de ECG digitales dentro de un SGBD de forma estandarizada, presentando tiempos de ejecución muy cercanos a los obtenidos por una aplicación ad-hoc para realizar búsqueda por semejanza sobre señales de ECG digitales.

1.4. Organización Restante del Trabajo

El resto del presente trabajo se estructura de la siguiente manera. En el Capítulo 2 se describen los trabajos encontrados en la literatura en el campo del procesamiento de señales de ECG digitales. Para la etapa de búsqueda por semejanza sobre señales de ECG digitales, se detallan los métodos y algoritmos propuestos. Se detalla también el uso en la literatura de los SGBD así como su extensión mediante el uso de UDFs. Tras la revisión de los trabajos en la literatura, se identifican las áreas de oportunidad que se presentan para este trabajo de tesis, resaltando principalmente la falta de soporte nativo para algunas funciones o tipos de datos por parte del SGDB para el tratamiento de señales de ECG digitales y la falta de estandarización para su almacenamiento.

En el Capítulo 3 se describen las técnicas y algoritmos utilizados para implementar búsqueda por semejanza dentro de un SGBD con la ayuda de UDFs. Posteriormente se describen nuestras propuestas para el almacenamiento de señales de ECG digitales

dentro de un SGBD, denominadas almacenamiento horizontal y almacenamiento vertical. También se describen los operadores identificados para implementar búsqueda por semejanza así como la estructura general que las consultas requieren para cada uno de los modelos de almacenamiento de los datos.

En el Capítulo 4 se describe la evaluación experimental donde se implementaron los operadores básicos requeridos para realizar búsqueda por semejanza sobre señales de ECG digitales en un SGBD comercial. Primeramente se describe el conjunto de datos de prueba utilizado para nuestra experimentación. Posteriormente se describe el tratamiento previo de los datos requerido para la identificación y preservación de la información de interés, con el fin de poder lograr su almacenamiento de manera estructurada. Después se detalla la implementación de los modelos de almacenamiento horizontal y almacenamiento vertical dentro del SGBD, así como los requerimientos de almacenamiento que cada uno conlleva. Por último en este capítulo, se detallan las consultas utilizadas durante nuestra etapa de experimantación, analizando posteriormente los tiempos de ejecución de cada una de ellas con el uso de distintos métodos de manejo de información generada entre las subconsultas que componen a una consulta estructurada.

Finalmente, en el Capítulo 5 se presentan las conclusiones de el presente trabajo, así como la formulación de interrogantes que pueden abrir camino hacia una investigación y trabajo futuro.

Capítulo 2
Estado del Arte

Las señales de electrocardiogramas (ECG) digitales han sido objeto de estudio por más de 100 años [18]. La primera grabación de la actividad eléctrica del corazón fue realizada en 1887 por el fisiólogo inglés August Waller, quien usó electrodos de superficie colocados en la piel de un paciente mediante un electrómetro capilar. A pesar de ello, se considera como padre de la electrocardiografía a Willem Einthoven, quien en 1902 grabó el primer ECG haciendo uso de un galvanómetro de cuerda. Desde entonces y hasta nuestra fecha, el tratamiento de señales de ECG digitales ha presentado un gran número de retos a resolver para su manejo y análisis.

Hoy en día, una de las líneas más predominantes en el campo del tratamiento de señales de ECG digitales es el procesamiento digital de las mismas. En esta área se han propuesto una gran cantidad de métodos y algoritmos con el fin de; a) mejorar la calidad de las señales de ECG digitales, b) detectar correctamente los latidos presentes en ellas, c) extraer las características propias de las señales de ECG digitales y, d) clasificarlas de manera eficiente [18, 56, 67]. A pesar de la extensa revisión de la literatura, hasta donde tenemos conocimiento no existe trabajo alguno que explícitamente haga uso exclusivo de un Sistema Gestor de Bases de Datos (SGBD) para el almacenamiento y administración de datos de señales de ECG digitales. Esto último representa el principal foco de atención del presente trabajo. Los trabajos encontrados en la literatura hacen referencia a bases de datos de secuencias [5, 35, 58, 66], bases de datos de series de tiempo [14, 16, 64] o simplemente bases de datos [20, 35]. A continuación de describen a detalle el marco referencial sobre el cual se apoya el presente trabajo.

2.1. Procesamiento de Señales de ECG Digitales

El procesamiento de señales de ECG digitales es el área de la ingeniería que se centra en la manipulación de señales de ECG con el objetivo de permitir la extracción de información de la señal por medios automatizados [59]. El procesamiento de una señal de ECG digital necesita (en la mayoría de los casos) realizar un gran número de cálculos. En particular, el procesamiento de señales de ECG digitales ha contribuido de manera significativa hacia el entendimiento del ECG y sus propiedades dinámicas, observadas por cambios en el ritmo cardíaco y en la morfología de sus latidos.

Cada señal de ECG digital contiene información obtenida a partir de la actividad eléctrica producida por las contracciones del corazón. Un ciclo cardíaco encontrado dentro de una señal de ECG consiste en una serie de latidos, también conocidos como ondas PQRST. Esta serie de ondas puede ser dividida en tres ondas básicas: P, QRS, y T, cada una de las cuales representa un fenómeno eléctrico específico encontrado dentro de la señal de ECG. La onda P representa la despolarización auricular. La onda QRS (también conocido como complejo QRS) representa la despolarización ventricular. Por su parte, la onda T representa la repolarización ventricular [59]. En un ciclo cardíaco normal, la onda P ocurre primero, seguida por el complejo QRS y la onda T. Las secciones entre las ondas presentes en las señales de ECG y los complejos se denominan segmentos. Cada señal de ECG se caracteriza por tres segmentos denominados: segmento PR, segmento ST y segmento TP, cuya ubicación se da entre las ondas que componen sus respectivos nombres.

En la literatura científica se definen una serie de etapas inherentes al procesamiento de señales de ECG digitales, y que se muestran en la Figura 2.1. Para cada una de las etapas se ha propuesto una gran cantidad de algoritmos. Sin embargo, el presente trabajo de tesis se centra en la etapa de búsqueda por semejanza, marcada en la Figura 2.1.

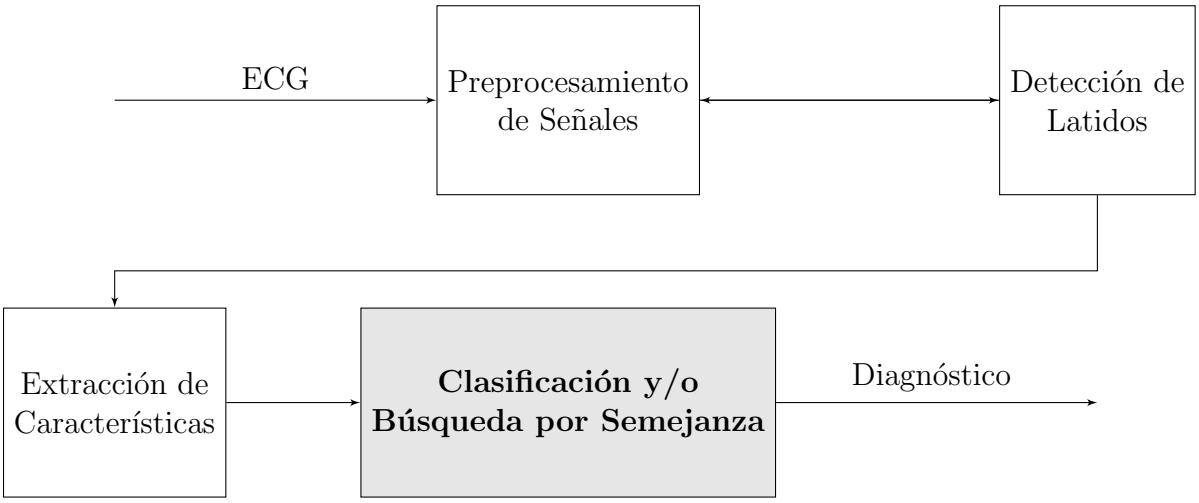


Figura 2.1: Etapas del Procesamiento de Señales de ECG Digitales

2.1.1. Etapas Previas a la Búsqueda por Semejanza en el Procesamiento de Señales de ECG Digitales

La primera etapa del procesamiento de señales de ECG digitales es conocida como preprocesamiento ó en algunos casos también llamada supresión de ruido [4, 18, 55, 29]. Especial atención debe debe ser prestado a esta etapa, debido a que necesita ser aplicada solamente cuando la información de interés no se distorsione tras su aplicación [55]. Típicamente, la entrada que reciben los algoritmos de esta etapa son señales de ECG digitales en su mas puro estado, es decir, no han sido tratadas previamente. Existen altas probabilidades (si no es que en todos los casos) de que cada señal de ECG digital estará siempre contaminado por varios tipos de ruido [18, 55]. Los tipos de ruido más comunes se describen a continuación.

- Ruido por contacto de electrodos: Es interferencia transitoria provocada por la pérdida de contacto entre los electrodos y la piel. La duración de este tipo de ruido suele ser de 1 segundo y la amplitud suele ser la máxima que se registra en toda la señal.

- Ruido por movimiento: Son cambios transitorios en la onda base de la señal

causados por impedancia entre electrodos y la piel. La amplitud de las ondas y la duración de este tipo de ruido suelen variar. La duración promedio ronda los 400-500ms con una amplitud 5 veces mayor que lo normal.

- Ruido por contracciones musculares: Este ruido se genera por las contracciones musculares por parte del paciente, causando variaciones a un nivel potencial mili voltaico. La variación en amplitud suele estar entre el 5-10 % más de lo normal y su duración ronda los 40-50ms.

- Curvatura de la onda base: Esta alteración causa problemas en la detección de latidos. La alteración a baja frecuencia suele ser causada por la respiración o por movimientos del paciente. La variación de amplitud causada es del 15 %. Este ruido debe ser removido del ECG antes de la extracción de características.

Una señal de ECG digital libre de ruido nos ofrece información detallada sobre la electrofisiología de los problemas del corazón y cambios que en él podrían ocurrir. Para poder ofrecer un soporte en la toma de decisiones clínicas, las señales de ECG digitales deben ser claramente comprensibles y libres de todo ruido [55, 51].

La segunda etapa en el procesamiento de señales de ECG digitales es la detección del complejo QRS. La presencia de un latido y el tiempo de su ocurrencia es información básica requerida para el resto de las etapas en el procesamiento de señales de ECG digitales. Dado que el complejo QRS es la onda más notable dentro de la señal del ECG, la etapa de detección de latidos es sinónimo de la detección de complejos QRS [59]. Un algoritmo para la detección de complejo QRS debe ser capaz de reconocer una gran variedad de morfologías QRS para que este sea aceptado clínicamente. La mayoría de los algoritmos para la detección de complejo QRS encontrados en la literatura han sido desarrollados con un enfoque *ad hoc* y validaciones experimentales. Para lograr detectar correctamente el complejo QRS dentro de la señal de ECG digital es necesaria la aplicación de un preprocesamiento. La idea de este paso previo es resaltar el complejo QRS mientras se suprimen los ruidos presentes en la señal

[69]. Una vez que la señal ha sido preprocesada y (opcionalmente) caracterizada, la señal es analizada utilizando una ventana deslizante. Una ventana deslizante es una estructura que permite analizar todas las posibles subsecuencias de longitud n' dentro de una señal de tamaño n. Junto con ello, se establece un conjunto de reglas de decisión para determinar si el complejo QRS se encuentra o no en la ventana actual. Estas reglas pueden ser tan sencillas como aplicar un umbral de amplitud, pero puede aplicarse también una comprobación de la forma de la onda [59].

La tercera etapa en el campo del procesamiento de señales de ECG digitales es la extracción de características. En ella existe un interés interdisciplinario en la representación de las señales para su procesamiento, almacenamiento y posterior análisis. Esto es atendido por el área de transformación de señales, también conocida como caracterización o extracción de características [16, 30, 64, 67]. Una característica es una medida distintiva, una transformación o un componente estructural extraído de una secuencia [55]. De esta manera, el objetivo de la etapa de extracción de características de una señal de ECG digital es encontrar tan pocas propiedades como sea posible dentro de la misma que permitan detectar anormalidades en las ondas correctamente y un pronóstico correcto.

En los últimos años se han propuesto métodos para extraer las ondas P-QRS-T de señales de ECG digitales. Cada onda (o combinación de ondas) corresponde a algún padecimiento cardíaco [25]. La onda P, por ejemplo, puede ser normal, inusualmente alta o inusualmente amplia. El complejo QRS puede tener solamente tres anormalidades: puede ser muy amplio, muy alto o contener una onda Q anormal. El segmento ST puede ser normal, elevado o decaído.

2.1.2. Clasificación y Búsqueda por Semejanza

La cuarta etapa en el campo del procesamiento de señales de ECG digitales está compuesta por la clasificación y búsqueda por semejanza. En los últimos años, la clasificación automática de señales de ECG digitales ha recibido gran atención por parte de la comunidad de ingeniería biomédica [62]. La clasificación asistida por computadora puede lograr altos niveles de exactitud y con ello lograr una mayor seguridad en los diagnósticos clínicos. Para que un sistema de clasificación de señales de ECG digitales logre un alto nivel de eficacia es necesario encontrar las formas características de las ondas que lo componen que discriminen efectivamente entre las categorías de la clasificación. Típicamente, cada latido dentro de una señal de ECG digital es caracterizado mediante distintas mediciones tales como la magnitud, duración, amplitud y rango entre otras. Dichas características son abstraídas dentro de un vector característico que puede ser utilizado para la clasificación de los latidos asignándoles comúnmente una categoría que corresponde a latidos normales y latidos anormales [55, 52].

Clasificación

Una gran variedad de métodos se han propuesto para la clasificación de señales de ECG digitales. Saxena *et al.* [54] utilizan redes neuronales artificiales para la clasificación de señales. Tras aplicar diversas variaciones en el número de capas ocultas y número de nodos en cada una de ellas, la mejor topología de la red para la clasificación de señales consistió en dos capas ocultas con cuatro nodos en cada una de ellas. Las características que toma como entrada el clasificador se basan en la amplitud, pendiente y duración de los latidos encontrados dentro de cada intervalo RR presentes en las señales.

Zhao *et al.* [70] proponen el uso de máquinas de vectores de soporte con un kernel Gaussiano para clasificar diferentes latidos presentes en señales de ECG digitales. El

vector característico de cada latido se compone de 32 coeficientes $\{< w_1, w_2..., w_{32} >\}$ obtenidos tras una transformación de onduletas Daubechies, concatenado del vector de cuatro coeficientes $\{< a_1, a_2, a_3, a_4 >\}$ obtenidos de un modelo de auto regresión. Por último, PCA es usado para reducir la dimensión de 36 a solo 3 dimensiones, generando el vector de entrada para el clasificador.

Tadejko y Rakowski [61] proponen el uso de morfología matemática en la etapa de extracción de características para evaluar el rendimiento de un clasificador automático de latidos anormales. El clasificador fue desarrollado utilizando mapas auto organizados y algoritmos de aprendizaje de cuantificación vectorial sobre las características obtenidas de las señales de ECG digitales. Cada señal dentro de una ventana de 550ms es convertida a un vector característico de 20 dimensiones utilizando operadores de erosión, dilatación, cerradura y apertura. El intervalo RR también se utiliza como una característica para formar finalmente un vector de 21 dimensiones que el clasificador toma como entrada.

Búsqueda por Semejanza

En el campo de búsqueda por semejanza, dado un patrón q denominado secuencia consulta, se trata de encontrar dentro de un conjunto de secuencias S a todas aquellas secuencias s_i o subsecuencias s'_i consideradas suficientemente semejantes a q dada una función de semejanza $Sem(s'_i, q)$ [5, 15, 66].

Típicamente los algoritmos utilizados para la búsqueda por semejanza se enfocan en encontrar las k secuencias s_i más similares a q, o bien todas aquellas secuencias s_i que tienen una semejanza con q por encima de un umbral de semejanza, ϵ. Esto dos enfoques son enmarcados dentro de dos problemáticas conocidas como vecinos más cercanos (K-NN) [14, 35, 39, 58, 66] y búsquedas por rango (ϵ-search) [5, 16, 44], respectivamente.

La búsqueda por semejanza en el caso de secuencias puede ser dividida en dos categorías distintas: Búsqueda de secuencias completa y búsqueda de subsecuencias.

En la búsqueda de secuencias completa, se trata de encontrar todas las secuencias s_i consideradas suficientemente semejantes a q. En este caso, s_i y q tienen la misma longitud. En la búsqueda de subsecuencias, se trata de encontrar todas aquellas secuencias s_i tal que contengan alguna subsecuencia s_i' que se considere suficientemente semejante a q, así como las posiciones donde se encuentra la mayor semejanza. En este caso s_i' y q deben tener la misma longitud. Cabe mencionar que la búsqueda de subsecuencias es una generalización de la búsqueda completa [44].

Agrawal *et al.* [5] proponen un algoritmo para solucionar el problema de ϵ-search, realizando una búsqueda de secuencias completa. En el algoritmo, cada secuencia s_i de longitud n es representada con los primeros f coeficientes de la transformada discreta de Fourier ($f \leq n$). El vector f-dimensional resultante es almacenado e indexado en una base de datos utilizando un método de indexación denominado R*-tree. Después, la secuencia de consulta q es también transformada a un punto f-dimensional y se busca un conjunto de secuencias candidatas mediante el uso del método R*-tree. Dada entonces una secuencia de consulta q, el índice R*-tree es utilizado para construir un conjunto de secuencias candidatas, el cual consiste en todos aquellos vectores f-dimensionales que tienen una semejanza con respecto a q sobre el umbral de semejanza ϵ-tree. El algoritmo de obtención de candidatos garantiza que no habrá falsas destituciones, es decir, se garantiza que el algoritmo obtenga del índice todas aquellas secuencias cuya semejanza se encuentra por encima del valor del umbral. Sin embargo, el algoritmo puede causar falsas alarmas debido a que utiliza sólo f características en lugar de n. Por ello, para cada secuencia candidata obtenida, la secuencia original es leída de disco y se calcula de nuevo la semejanza con respecto a q utilizando la función de semejanza. Cada candidato que resulte ser una falsa alarma es descartado en un paso denominado etapa de post-procesamiento.

Faloutsos *et al.* [16] propusieron el primer algoritmo para la búsqueda de subsecuencias sobre la problemática ϵ-search como una generalización de la búsqueda de secuencias completa. En el algoritmo de construcción del íncide, este método divi-

de las secuencias s_i de longitud n en ventanas deslizantes de longitud w, generando $n-w+1$ subsecuencias. Cada una de las subsecuencias generadas es transformada en un punto f-dimensional que es almacenado individualmente con el uso de un índice denominado Índice-F. Faloutsos *et al.* utilizan rectángulos de delimitadores mínimos (MBR, *Minimum Bounding Rectangles*) que pueden contener cientos de puntos utilizando métodos heurísticos, y almacenan dichos MBR en el índice. Así, al especificar un patrón de búsqueda q y un umbral de semejanza ϵ, Faloutsos *et al.* dividen el patrón q en $Len(q)/w$ ventanas (a las cuales llaman ventanas separadas) de tamaño w y realizan una búsqueda por rango y general el conjunto de sequencias candidatas utilizando el índice-F y la distancia Euclideana como función de semejanza. Finalmente, aplican la etapa de post-procesamiento para descartar falsas alarmas.

En el trabajo de Moon *et al.* [45] se propone un nuevo método para la búsqueda de subsecuencias que busca resolver los problemas en el trabajo de Faloutsos *et al.* [16], utilizando de manera similar algoritmos para la construcción de índices y para la búsqueda de subsecuencias similares. Este método se denomina *Dual Match*. Dual Match se basa en el trabajo de Faloutsos *et al.* para la representación de las subsecuencias de s_i como ventanas separadas y la secuencia de consulta q en ventanas deslizantes. De esta manera, Dual Match reduce el número de puntos que se almacenan en una razón de $1/w$ veces con respecto al trabajo de Faloutsos *et al.* Por lo tanto, Dual Match es capaz de almacenar puntos individuales en lugar de conjuntos de puntos en el índice, utilizando un menor espacio en disco que Faloutsos *et al.*. Para la búsqueda de subsecuencias, Dual Match divide la secuencia de consulta en ventanas deslizantes de longitud w y las transforma en un vector f-dimensional con el uso de un índice R*-tree. Con ello se crea un conjunto de secuencias candidatas comparando cada punto que se encuentra en el índice contra los vectores f-dimensionales obtenidos de q. Finalmente, aplica la etapa de post-procesamiento para descartar falsas alarmas.

General Match [44] fue después propuesto por Moon *et al.* Este método busca resolver el problema del incremento en el número de falsas alarmas cuando se utilizan ventanas deslizantes de longitud corta encontrado en Dual Match. General Match se enfoca en la búsqueda de subsecuencias bajo la problemática de ϵ-search. General Match consiste en tres etapas, 1) inicialización, 2) búsqueda en el índice y 3)post-procesamiento. En la etapa de inicialización, se construyen MBRs que se usarán en la búsqueda con ayuda de un índice R*-tree. Las secuencias s_i son divididas en ventanas deslizantes generalizadas denominadas j-ventanas deslizantes y cada ventana es transformada en un punto f-dimensional. Con los puntos generados por cada secuencia s_i se crea el MBR y se almacena en el índice. En la etapa de búsqueda en el índice la secuencia de consulta q es dividida en ventanas separadas generalizadas denominadas j-ventanas separadas, y cada una de ellas es transformada en un punto f-dimensional. Con ello se construye un conjunto de secuencias candidatas utilizando cada MBR almacenado y cada punto obtenido de q, utilizando una función de semejanza y un valor de umbral de semejanza ϵ. Finalmente, en la etapa de post-procesamiento se seleccionan aquellas subsequencias similares descartando las falsas alarmas que se encuentran en el conjunto de candidatos.

López *et al.* [39] proponen el uso de funciones definidas por el usuario (UDFs) para implementar la búsqueda de subsecuencias dentro de un sistema gestor de bases de datos (SGBD). Un SGBD es un software especializado en la creación, administración y consulta de grandes conjuntos de datos de manera eficiente [19, 53]. Una UDF forma parte del estándar SQL, y se define como una subrutina escrita en un lenguaje de programación de alto nivel (C o C++, por ejemplo) [49]. Una vez escrita, la UDF es compilada y después enlazada con el propio SGBD. De esta manera, la UDF puede ser llamada mediante la utilización de una consulta de tipo SELECT [49]. López *et al.* almacenan las señales de ECG digitales de dos maneras distintas, como archivos de texto plano y como BLOBs (Binary Large Objects). Sin embargo, ambas

propuestas de almacenamiento carecen de operadores nativos por parte del SGBD para su tratamiento, por lo que se basa completamente en el uso de una UDF a la medida para realizar búsqueda por semajanza sobre cada cada uno de los modelos de almacenamiento utilizados. El trabajo de López *et al.* concluye que es posible extender la funcionalidad del SGBD utilizando UDFs para realizar búsqueda por semejanza en señales de ECG digitales. De igual manera su evidencia experimental demuestra que los tiempos de ejecución necesarios con el uso de BLOBs se aleja significativamente del tiempo promedio requerido por aplicaciones ad-hoc. Por otro lado, el uso de archivos de texto plano presenta tiempos de ejecución muy cercanos a los tiempos necesarios por una aplicación ad-hoc. Sin embargo, con ello su propuesta sigue dependiendo directamente del sistema de archivos del sistema operativo, lo que conlleva a riesgos que no pueden ser controlados por parte del SGBD.

2.2. Sistemas Gestores de Bases de Datos

La cantidad de datos disponible para manejar en nuestros tiempos se está incrementando de manera constante. Para llegar a obtener el mejor provecho de grandes y complejos conjuntos de datos se requiere de herramientas que simplifiquen las tareas para su tratamiento y análisis en el menor tiempo posible, así como para su almacenamiento [53].

Una manera de aprovechar y manejar el potencial de las bases de datos es utilizando un software denominado Sistema Gestor de Bases de Datos (SGBD) [19]. Un SGBD es un software especializado en la creación, administración y consulta de grandes conjuntos de datos de manera eficiente [19, 53]. Los primeros SGBD aparecieron a finales de los años 60's, y revolucionaron la manera de manejar y almacenar datos.

En el año 1970, Edgar F. Codd propuso un nuevo marco para la representación de los datos denominado *modelo de datos relacional* [11]. Codd estableció que los SBGD debían presentar al usuario una vista de datos organizada a manera de tablas,

denominadas *relaciones*, además de otros conceptos principales como *atributos*, *tuplas*, *esquemas*, *dominios*, entre otros, y que son detallados en su trabajo. Con esta representación de datos, las consultas de datos podrían ser expresadas en un lenguaje de muy alto nivel. De esta manera se dio lugar a la aparición de los Sistemas Gestores de Bases de Datos Relacionales (SGBDR), basados en el modelo relacional propuesto por Edgar F. Codd y que sigue vigente hasta nuestros días. Al utilizar un SGBDR, el uso de relaciones es inherente a la aplicación. Dentro del campo de los SGBD, hacemos referencia a una relación como la unidad básica para la representación de datos [40]. Una relación está compuesta por uno o más atributos. Cada atributo se encuentra asociado a un tipo de dato (entero, cadena, decimal, etc.). Dentro de una relación, los datos son almacenados como tuplas. Como resultado, una relación puede parecerse a una tabla, donde los atributos representan elementos individuales de los datos y las tuplas identifican registros completos de los datos. En la Figura 2.2 se puede observar la estructura básica de una relación denominada "Personas". La relación "Personas" tiene cuatro atributos y tres tuplas. El primer atributo llamado "ID" tiene asignado un tipo de dato entero. El segundo atributo, denominado "Nombre" es de tipo cadena de caracteres. El tercer atributo, "Edad" tiene también asignado un tipo de dato entero. Finalmente, el cuarto atributo, denominado "Sueldo" tiene un tipo de dato de punto flotante. Las tuplas hacen referencia a tres personas diferentes y la descripción de sus características. Una relación consta también de un *esquema* y una *instancia*. El esquema de la relación especifica el nombre de la misma, el nombre de cada atributo, y el dominio de cada atributo. La instancia de una relación es un conjunto de tuplas, cada una con el mismo número de atributos especificados en el esquema de la relación [40, 53].

Los beneficios del modelo relacional han sido ampliamente reconocidos, a tal punto de que el uso de SGBD relacionales para el manejo de datos se ha mantenido como una práctica hasta nuestros tiempos. Algunos de los beneficios brindados por el modelo relacional se listan a continuación [53].

ID	Nombre	Edad	Sueldo
1	Marcos Aguilar	23	5,234.5
2	Juan Esquerra	42	4,420.9
3	Luis Ibarra	25	4,928.7

Figura 2.2: Representación Gráfica de una Relación llamada “Personas”.

- Independencia de Datos: El SGBD provee una vista abstraída de los datos, la cual oculta a otras aplicaciones los detalles de representación y almacenamiento físico de los datos.
- Acceso Eficiente a Datos: El SGBD utiliza técnicas sofisticadas para el almacenamiento y consulta de datos de manera eficiente.
- Integridad de Datos: Mientras los datos sean accesados mediante el SGBD, este puede imponer restricciones de integridad, lo que le permite asegurar que los datos se apegan a la realidad y se encuentren actualizados.
- Acceso Concurrente y Recuperación de Fallos: El SGBD permite el acceso a varios usuarios de manera paralela.
- Atomicidad: El SGBD garantiza transacciones completas (conjunto de órdenes que se ejecutan como una única orden), que delimitan a una operación, o bien a realizarse completamente, o a no realizarse.
- Consistencia: El SGBD asegura que ante la ejecución de una transacción, la base de datos es llevada siempre de un estado válido a otro estado válido.
- Persistencia: El SGBD asegura que una vez realizada una operación, ésta persistirá (se mantendrá guardada) y no podrá deshacerse incluso aunque el sistema mismo falle.

En el año 2005, Michael Stonebraker presentó el diseño para un nuevo SGBD optimizado para la realización de consultas analíticas (OLAP, On-Line Analytical Processing) denominado *C-Store* [57]. En ese entonces, la mayoría de los SGBD estaban implementados como sistemas orientados a transacciones (OLTP, On-Line Transaction Processing). Entre las principales diferencias que presentó *C-Store* con respecto a los SGBD existentes se encuentran: el almacenamiento físico de datos por atributos y no por tuplas, uso de técnicas de ordenamiento y compresión de valores que reducen los requerimientos de almacenamiento y por último, el reemplazo de tablas e índices por una colección de proyecciones columnares. *C-Store* mantiene el soporte para el modelo lógico relacional de datos. En consecuencia, el lenguaje de consulta para la información almacenada en *C-Store* continúa siendo SQL, el lenguaje de consulta estándar dentro de un SGBD [46].

En la actualidad, los SGBD se enfocan en la optimización de operaciones de entrada/salida de datos a disco [57]. Un bloque de disco representa la unidad mínima de almacenamiento presente en un dispositivo de almacenamiento. En los SGBD especializados en la ejecución de consultas de tipo OLTP (y a los cuales a lo largo de este trabajo nos referimos como *SGBD Row-Store*), pocas escrituras a disco bastan para almacenar toda la información referente a una tupla, debido a que usualmente almacena la información de una misma tupla dentro de un mismo bloque de disco (o en su defecto, un conjunto de bloques de disco). Por esta razón, se obtiene un gran rendimiento en escritura, y se le denomina a dicho SGBD un sistema optimizado para escritura [57]. Por otro lado, un SGBD orientado a columnas (y al cual a lo largo de este trabajo nos referimos como *SGBD Column-Store*), está especializado en la ejecución de cargas de trabajo que requieren de muchas lecturas a disco sobre pocos atributos. Este comportamiento se presenta gracias a que se almacena toda la información referente a un mismo atributo dentro de un mismo bloque de disco, evitando la lectura de bloques que contienen datos de atributos innecesarios en la consulta. Por esta razón a los *SGBD Column-Store* se les denomina como sistemas optimizados

ID	C1	C2	C3	C4
1	V11	V12	V13	V14
2	V21	V22	V23	V24
3	V31	V32	V33	V34

(a) Representación Lógica de los Datos en un *SGBD Row-Store*.

B1	1	V11	V12	V13
	V14			
B2	2	V21	V22	V23
	V24			
B3	3	V31	V32	V33
	V34			

(b) Representación Física de los Datos en un *SGBD Row-Store*.

Figura 2.3: Almacenamiento en un *SGBD Row-Store*.

para la ejecución de consultas OLAP [57].

2.2.1. Sistemas Gestores de Bases de Datos Row Store y Column Store

Históricamente, la implementación y desarrollo sobre los SGBD se han enfocado en los *SGBD Row-Store* [1, 57]. El éxito de estos sistemas se ha visto agrandado gracias a que presentan un buen desempeño ante las aplicaciones más comunes para bases de datos (consultas OLTP). En un *SGBD Row-Store*, los datos de una relación son almacenados tupla por tupla en los bloques de disco. De esta manera, los datos de una tupla se almacenan a nivel físico de manera contigua. La Figura 2.3 ilustra como es almacenada una relación lógica 2.3a y físicamente 2.3b compuesta por cinco atributos y tres tuplas dentro de un *SGBD Row-Store*, utilizando dos bloques de disco etiquetados como **B1** y **B2**.

Gracias al hecho de que cada vez emergen más aplicaciones que requieren de análisis, planeación e inteligencia de negocios, ha surgido un mayor interés en el desarrollo de nuevos trabajos que comparan el rendimiento de *SGBD Column-Store* y *SGBD Row-Store* [1, 2, 24, 57]. En un *SGBD Column-Store* los datos de un mismo atributo son almacenados de manera contigua en un bloque de disco. En la Figura 2.4 se ilustra como es almacenada una relación compuesta por cinco atributos y tres tuplas

ID	C1	C2	C3	C4
1	V11	V12	V13	V14
2	V21	V22	V23	V24
3	V31	V32	V33	V34

(a) Representación Lógica de los Datos en un *SGBD Column-Store*.

B1	B2	B3	B4	B5
1	V11	V12	V13	V14
2	V21	V22	V23	V24
3	V31	V32	V33	V34
⋮	⋮	⋮	⋮	⋮

(b) Representación Física de los Datos en un *SGBD Column-Store*.

Figura 2.4: Almacenamiento en un *SGBD Column-Store*.

dentro de un *SGBD Column-Store*, utilizando cinco bloques de disco etiquetados como **B1**, **B2**, **B3**, **B4** y **B5**.

2.2.2. Diferencias Entre Row Store y Column Store

En el trabajo de Abadi [1] se mencionan algunas de las diferencias de mayor importancia entre los *SGBD Column-Store* y los *SGBD Row-Store*, listadas a continuación.

- Las consultas OLAP operan sobre atributos más que sobre tuplas. Tanto en un *SGBD Row-Store* como en un *SGBD Column-Store* es necesario leer de disco aquellos bloques que contienen la información de los atributos presentes en la consulta. Sin embargo, en un *SGBD Row-Store* la cantidad de información leída de disco y que no es necesaria para procesar la consulta suele ser mayor, existiendo un desperdicio de lectura debido a que en el mismo bloque pueden existir atributos no requeridos por la consulta, sobre todo cuando una tupla tiene muchos atributos (cientos).

- Las relaciones con un gran número de atributos representan un gran reto para los *SGBD Row-Store*. Imaginemos que contamos con una relación, en la cual se tienen mil atributos y 500 mil tuplas. Imaginemos ahora que tenemos una consulta en la que se requiere calcular el promedio de un atributo y la sumatoria

de otro atributo. Para que un *SGBD Row-Store* ejecute la consulta, necesita leer de disco todos aquellos bloques que contienen los dos atributos que se necesitan. Con esta acción se leerán también todos aquellos atributos irrelevantes que se encuentren en el mismo bloque de disco. Un *SGBD Column-Store* por su parte, requiere leer de disco solo aquellos bloques que contengan los dos atributos que se necesitan, sin necesidad de traer consigo la información de atributos irrelevantes para la construcción del resultado.

- Cuando una nueva tupla es agregada a la base de datos, necesitan también ser agregados múltiples atributos. Para realizar la inserción de dichos atributos, un *SGBD Column-Store* necesita escribir distintos bloques de disco, uno para cada atributo. Un *SGBD Row-Store*, podría llegar a requerir solamente la lectura/escritura de un bloque de disco si la información de la tupla no sobrepasa el tamaño del bloque.

2.2.3. Sistemas Gestores de Bases de Datos Híbridos

En el trabajo de Abadi *et al.* [2] se ha demostrado que los *SGBD Column-Store* tienen un mejor desempeño que los *SGBD Row-Store* para las cargas de trabajo de naturaleza analítica. La principal razón de esto es la eficiencia que presentan los *SGBD Column-Store* en términos del número de operaciones de lectura necesarias ante consultas OLAP. Como resultado del rendimiento observado ha surgido la inquietud de saber si existe la posibilidad de obtener los beneficios de un *SGBD Column-Store* dentro de un *SGBD Row-Store*.

En la literatura se encuentran tres enfoques principales para la combinación de un *SGBD Column-Store* con un *SGBD Row-Store*. El primero de estos enfoques se basa en el uso de un *SGBD Row-Store* sobre el cuál se emula el comportamiento de un *SGBD Column-Store*, mediante una técnica denominada particionamiento vertical de tablas [32]. Este enfoque no necesita modificar el código del SGBD, solamente se

necesita de la modificación del esquema de las relaciones. El segundo enfoque realiza una modificación sobre el administrador de almacenamiento en disco para que almacene físicamente las relaciones atributo por atributo. El esquema de las tablas se mantiene intacto, y el motor de ejecución de consultas no necesita ser modificado. El tercer enfoque requiere de modificaciones tanto en el administrador de almacenamiento en disco para que escriba los datos de manera columnar, y del motor de ejecución de consultas para que lea y procese los datos también de manera columnar. De esta manera, se le brinda el funcionamiento de un *SGBD Column-Store* completamente a un *SGBD Row-Store*. La dificultad de implementación que presentan los enfoques mencionados anteriormente va de más fácil a más difícil respectivamente.

La diferencia en rendimiento de los tres enfoques ha sido revisada en el trabajo de Abadi *et al.* [2], en donde se comparó el rendimiento del SGBD orientado a columnas C-Store [57] contra distintos *SGBD Row-Store* comerciales. El trabajo concluye que es posible obtener algunas de las ventajas de un *SGBD Column-Store* implementado sobre un *SGBD Row-Store*. Sin embargo, para poder obtener la mayor cantidad de los beneficios que ofrece un *SGBD Column-Store*, se necesita realizar cambios tanto en el administrados de almacenamiento en disco como en el motor de ejecución de consultas, tal y como lo describe el tercer enfoque. Por lo tanto, los beneficios ofrecidos por cada uno de los enfoques anteriores, es inversamente proporcional a la dificultad de su implementación. Cabe mencionar entonces, que la construcción y/o modificación de un *SGBD Row-Store* que pueda simular el funcionamiento de un *SGBD Column-Store* donde el almacenamiento por atributos se desempeñe de manera eficiente representa un problema interesante para seguir siendo investigado.

2.3. Áreas de Oportunidad

La utilización de herramientas por separado para la gestión del almacenamiento de señales y análisis de información presenta serias desventajas que deben ser resueltas, principalmente la falta de estandarización para la representación de información no estructurada y los cuellos de botella existentes entre importación y exportación de datos. En este Capítulo se resalta la falta de información en la literatura que haga uso exclusivo de un SGBD para la gestión de datos de señales de ECG digitales utilizando una serie de operadores que a su vez puedan ser reutilizados y además ofrezca una solución cuyos tiempos de ejecución se aproximen a los de una aplicación ad-hoc. Esto nos da una idea inicial para la propuesta de nuestros modelos de almacenamiento y técnicas de acceso a los datos utilizando únicamente un SGBD para el tratamiento de señales de ECG digitales. En especial, para realizar búsqueda por semejanza de patrones en señales de ECG digitales.

Gracias a la facilidad ofrecida por los SGBD actuales, podemos extender su funcionalidad mediante el uso de UDFs. De esta manera, queda abierto un campo de oportunidad que sea capaz de combinar los beneficios que cada una de las líneas del conocimiento descritas en este capítulo nos ofrecen. Primeramente, la identificación de los métodos y algoritmos del procesamiento de señales de ECG digitales que pueden ser implementados dentro de las capacidades de los SGBD actuales. Una vez identificados, la propuesta de representación para las señales de ECG digitales dentro de un SGBD sería una gran aportación para establecer un estándar en su almacenamiento. Aunado a esto, son necesarias las consultas que se requieren para tratar y consultar la información almacenada, así como la implementación y vinculación de las funciones necesarias al SGBD y que deseablemente nos permitan obtener tiempos de ejecución aceptables.

Capítulo 3
Implementación de Operaciones para Realizar Búsqueda por Semejanza Sobre Señales de ECG Digitales

En este Capítulo se demuestra que es posible utilizar únicamente el Sistema Gestor de Bases de Datos (SGBD) para aplicar operadores que se encuentran dentro del campo de búsqueda por semejanza sobre señales de ECG digitales. Esto nos permite evitar el uso de herramientas adicionales y problemas de compatibilidad. Al mismo tiempo nos permite reducir los tiempos de espera entre la exportación e importación de datos. En la Figura 3.1 se ilustra la estructura general utilizada en este trabajo de tesis para lograr la implementación de un núcleo de operadores básicos en el campo del procesamiento de señales de ECG digitales dentro de un SGBD. En primer lugar se contempla el almacenamiento de las señales de ECG digitales dentro del SGBD, para lo cual es necesario definir los modelos de almacenamiento que se van a utilizar (Figura 3.1a).

En este trabajo de tesis se propone el uso de dos modelos lógicos de almacenamiento, denominados Almacenamiento Horizontal y Almacenamiento Vertical, respectivamente. De forma paralela al diseño de los modelos de almacenamiento (Figura 3.1b) se puede realizar la selección de algoritmos utilizados en campo de búsqueda por semejanza sobre señales de ECG digitales que se deseen implementar como funcionalidad del SGBD. Una vez que los algoritmos han sido seleccionados, el siguiente paso es identificar los operadores básicos que los componen. Específicamente, en este trabajo describimos el desarrollo de los operadores necesarios para una de las etapas identificadas en la literatura. Esta etapa corresponde a la búsqueda por semejanza, analizando las problemáticas de búsqueda por vecinos más cercanos (K-

NN) y búsqueda por rango (también conocida como ϵ-search) por su relevancia en la literatura científica [5, 15, 16, 20, 35, 66].

Una vez que los operadores básicos para implementar búsqueda por semejanza han sido identificados, el siguiente paso es implementarlos dentro del SGBD. Para ello, se pueden utilizar funciones proveídas por el propio SGBD si se adecuan a nuestras necesidades. Sin embargo, la falta de estandarización para el almacenamiento de señales de ECG digitales por parte de los SGBD actuales nos orilla a la implementación de nuestras propias funciones mediante mecanismos que nos permiten extender las capacidades del SGBD, denominados Funciones Definidas por Usuario (UDFs). Por último, una vez que los modelos de almacenamiento han sido definidos y los operadores básicos necesarios han sido identificados e implementados, se necesita describir la estructura general que las consultas SQL deben tener para cada uno de los modelos de almacenamiento propuestos. Cada una de estas consultas se encuentra a su vez estructurada por una serie de subconsultas representando los operadores identificados previamente. A continuación se describen a detalle cada uno de los pasos seguidos en este trabajo de tesis.

3.1. Propuesta de Modelos de Almacenamiento

Una señal de ECG digital puede ser abstraída como una serie de tiempo, la cual a su vez, omitiendo el índice que representa el tiempo puede ser representada como una secuencia de valores reales de la forma $s = (s[0], [1], ..., s[n])$, donde $s[n] \in \mathbb{R}$ [13]. Partiendo de la definición de secuencia, a continuación se detalla la definición y características de los SGBD actuales, seguido por nuestras propuestas para el almacenamiento de señales de ECG digitales dentro de un SGBD comercial.

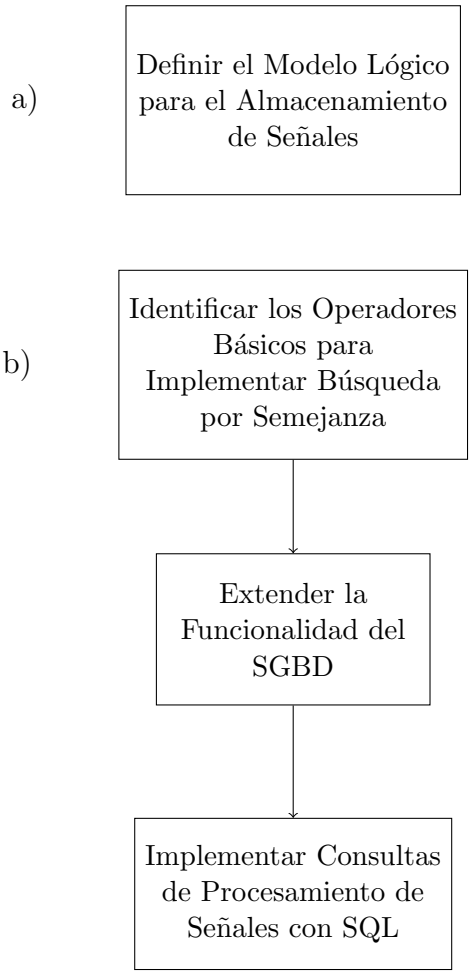


Figura 3.1: Pasos Necesarios para Realizar Algoritmos de Búsqueda por Semejanza Sobre Señales de Electrocardiogramas Digitales Dentro de un Sistema Gestor de Bases de Datos.

3.1.1. Almacenamiento Horizontal

Para nuestra primera propuesta, denominada almacenamiento horizontal, una relación S_H almacena señales de ECG digitales s_i abstraídas como parte de una tupla. Dicha tupla tiene la forma $T_{Hi} = (i, s_i)$, donde i representa el identificador de la señal y s_i representa a cada señal como un atributo de S. De esta manera, todas las señales s_i son almacenadas dentro de S_H de la forma $S_H = \{T_{Hi}\}$ donde $1 \leq i \leq N$. Así, el esquema de la relación S_H para nuestra propuesta de almacenamiento horizontal es:

$$S_H < id, ecg >$$

Donde *id* representa un identificador único de una señal y *ecg* representa la señal de ECG digital almacenada.

3.1.2. Almacenamiento Vertical

Para la implementación de nuestra segunda propuesta, denominada almacenamiento vertical, una relación S_V almacena N señales de ECG digitales s_i abstraídas como un conjunto de tuplas de la forma $T_{Vi} = \{(i, j, s_i[j])\}$, donde i representa el identificador de la señal, y $s_i[j]$ representa el valor de s_i en la j-ésima posición. De esta manera, se almacenan $n \times N$ tuplas en S_V. Cada una de las señales s_i puede ser obtenida de manera individual mediante todas aquellas tuplas con un mismo identificador i y ordenadas posteriormente por índice j. Con esto, el esquema de la relación S_V para el almacenamiento vertical es:

$$S_V < id, indice, valor >$$

Donde *id* representa un identificador único de cada señal, *indice* representa la posición dentro de la señal y *valor* representa propiamente el valor almacenado en dicha posición.

3.2. Desarrollo de Algoritmos de Búsqueda por Semejanza

La búsqueda por semejanza es un área en crecimiento dentro del procesamiento de señales de ECG digitales [5, 15, 16, 20, 35, 66]. Esta área se divide en dos partes principales, denominadas búsqueda completa y búsqueda de subsecuencias. Estas dos últimas se dividen a su vez en búsquedas denominadas *Top-k* y búsquedas por rango, las cuales son mejor conocidas como las problemáticas K-NN y ϵ-search respectivamente, y cuyos algoritmos desarrollados son descritos en las siguientes secciones. La Figura 3.2 ilustra la división de áreas a partir de la búsqueda por semejanza. En el

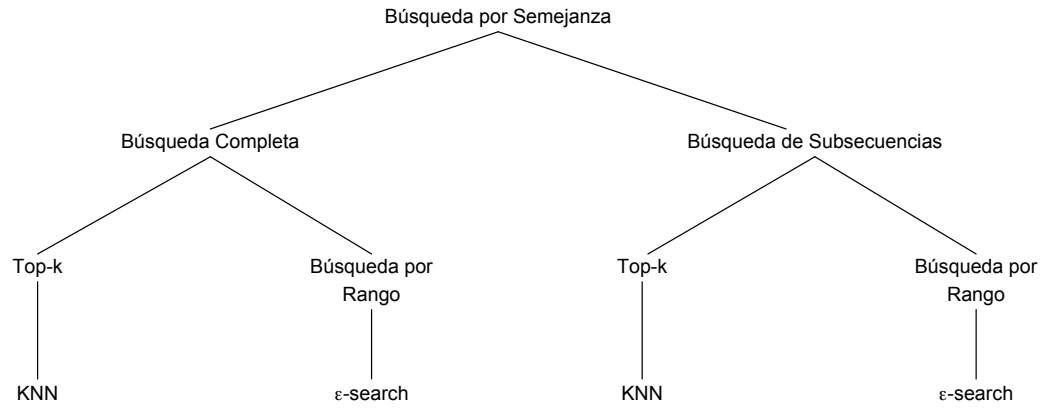


Figura 3.2: División de Áreas en el Campo de Búsqueda por Semejanza.

resto de este trabajo nos enfocamos en la descripción y desarrollo de la búsqueda de subsecuencias bajo las problemáticas K-NN y ϵ-search sobre señales de ECG digitales almacenadas dentro de un SGBD.

Dado un conjunto de señales de ECG digitales $S = \{s_1, s_2, ..., s_N\}$, con señales de la forma $s_i = (s_i[0], s_i[1], ..., s_i[n_i - 1])$, una subsecuencia de s_i es definida como $s'_i = s_i[p : l]$, la cual comprende los valores que se encuentran en s_i desde el índice p hasta l, siendo $(l-p)+1$ la longitud de s'_i. Dado un conjunto de patrones $Q = \{q_1, q_2, ..., q_M\}$, para cada patrón $q_j = (q_j[0], q_j[1], ..., q_j[m_j - 1])$, la búsqueda de subsecuencias se encarga de encontrar tripletas de la forma $(i, p, Sem(s'_i, q_j))$, donde i es el identificador de la señal s_i y, p es la posición donde se encuentra la subsecuencia s'_i considerada suficientemente semejante a q_j mediante alguna función de semejanza $Sem(s'_i, q_j)$ (Sección 3.2.1). Para poder obtener las subsecuencias $s'_i \in s_i$ nos enfocamos en una búsqueda exhaustiva dentro de s_i para calcular la semejanza con respecto a q_j. Es decir, para cada señal s_i se extraen $(n - m) + 1$ subsecuencias s'_i contra las cuales se calcula la semejanza respecto a q_j. De esta manera se genera una serie de semejanzas $D = (d_1, d_2, ..., d_{(n-m)+1})$, de donde posteriormente se obtiene la información necesaria para que los algoritmos propuestos tanto para resolver K-NN como ϵ-search regresen sus respectivos resultados.

La propuesta realizada en nuestro trabajo tiene como objetivo demostrar que es posible realizar el procesamiento de señales de ECG digitales dentro de un SGBD. El análisis y la propuesta de algoritmos eficientes en el campo del procesamiento de señales de ECG digitales es un tema sin duda interesante. Sin embargo, por razones de tiempo se encuentra fuera del alcance de este trabajo de tesis el desarrollo de la solución óptima. De esta manera, exponemos el escenario más complejo que podemos atacar, y con ello, se da como base una propuesta que pueda ser mejorada en trabajos futuros.

3.2.1. Medidas de Semejanza

Una medida de semejanza cuantifica la diferencia en términos de forma o distancia entre dos secuencias [7]. Formalmente, una medida de semejanza es una función $d(s_1, s_2)$, que toma como argumento dos secuencias s_1 y s_2, y regresa un valor numérico:

$$d(s_1, s_2) : \tau \times \tau \rightarrow \mathbb{R}$$

Una medida de semejanza es el componente principal para muchos algoritmos de clasificación y búsqueda [26, 66]. El valor numérico devuelto por $d(s_1, s_2)$ se interpreta como un índice de semejanza entre s_1 y s_2. De manera intuitiva, entre menos semejantes son s_1 y s_2, el valor de $d(s_1, s_2)$ es más grande, sin embargo, el sentido que se le da a dicho valor depende totalmente de el algoritmo que se utilice para el cálculo de semejanza. Las medidas de semejanza más prominentes en la literatura científica para secuencias incluyen: *Distancia Euclideana* [5, 8, 15, 16, 28, 35, 37, 38, 39, 63, 68], *Distorsión Dinámica de Tiempo* (DTW, Dynamic Time Warping) [31, 67] e *Índice de Correlación Cruzada* [10, 33]. A continuación se describen dichas medidas se semejanza. Nuestra metodología es capaz de incluir cualquier medida de semejanza para el cálculo de $d(s_1, s_2)$, y poder ser implementada como una función propia del SGBD (una UDF).

- **Distancia Euclideana**: La distancia Euclideana entre dos secuencias s_1 y s_2, ambas con la misma longitud n, se describe como la raíz cuadrada de la sumatoria de las diferencias al cuadrado entre el j-ésimo valor de s_1 y el j-ésimo valor de s_2, como se observa en la Ecuación 3.1.

$$D_{euc}(s_1, s_2) = \sqrt{\sum_{j=0}^{n-1}(s_1[j] - s_2[j])^2} \tag{3.1}$$

  El resultado de D_{euc} es un valor entre 0 e ∞, donde 0 (cero) indica que s_1 es idéntico a s_2. En la Figura 3.3 se ilustran tres secuencias obtenidas de tres señales diferentes de ECG digitales, denominadas S_1, S_2 y S_3 con una misma longitud n. Es importante mencionar que la distancia Euclideana es sensible a las unidades de medida de las variables, por lo que tanto s_1 como s_2 deben estar sobre la misma escala, tal es el caso de S_1, S_2 y S_3 en la Figura 3.3. La distancia Euclideana también es sensible al desplazanmiento de los valores, como sucede con S_1 y S_2 en la Figura 3.3. El cálculo de semejanza entre S_1, S_2 y S_3 utilizando distancia Euclideana es realizado punto a punto, tal y como se ilustra en la Figura 3.4. El resultado de distancia Euclideana establece que las secuencias más semejantes son S_1 y S_3 mientras que el valor semejanza entre S_1 y S_2 es menor a pesar de que sus formas son visualmente más semejantes (Figura 3.7a).

- **Índice de Correlación Cruzada** (ICC): Dadas dos secuencias s_1 y s_2, ambas de longitud n, el $ICC(s_1, s_2)$ (Ecuación 3.2) corresponde al cálculo de la covarianza de s_1 y s_2 (Ecuación 3.3), entre la raíz cuadrada del producto de la varianza de s_1 por la varianza de s_2 (Ecuación 3.4), Donde μ_s corresponde a la media aritmética de s. A diferencia de la distancia Euclideana, ICC se beneficia de la correlación existente entre las variables de las secuencias. A pesar de que las amplitudes de s_1 y s_2 se encuentren en escalas diferentes, ICC se es capaz de comparar la morfología de las secuencias más que la distancia entre sus valores

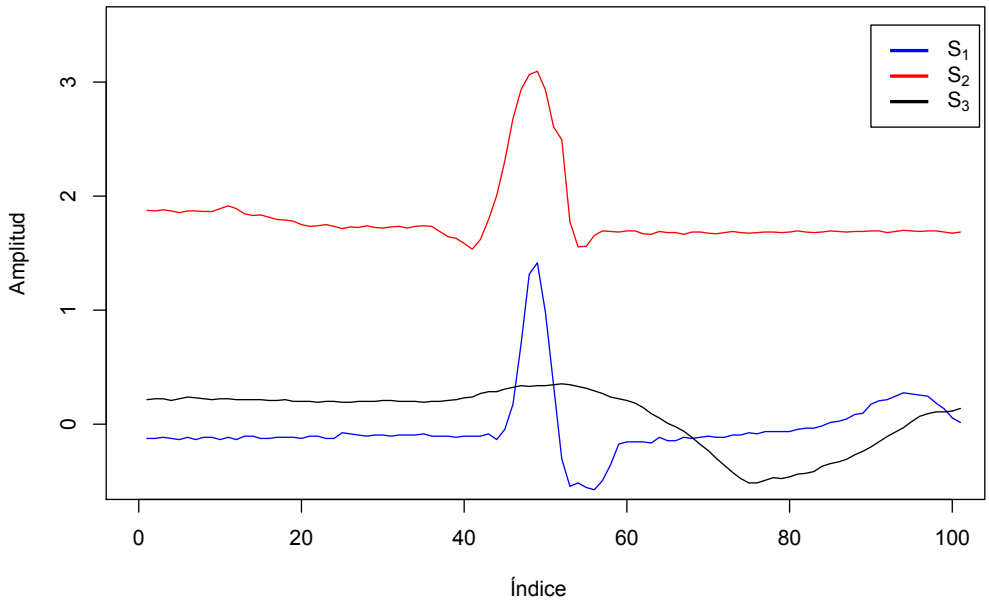


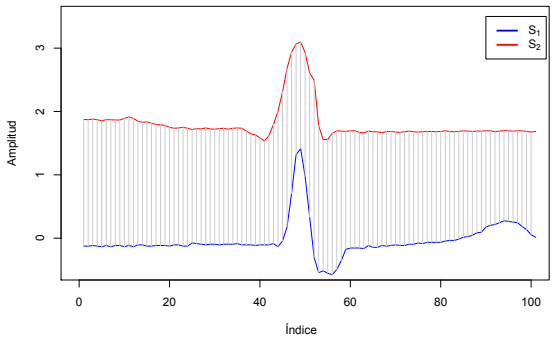
Figura 3.3: Tres Secuencias Denominadas S_1,S_2 y S_3 de Misma Longitud $n = 100$.

[10]. El resultado de $ICC(s_1, s_2)$ es un valor entre -1 y 1, denotando el sentido en el que s_1 y s_2 están correlacionadas. Un valor de 1 indica que cuando un valor $s_1[i]$ incrementa o decrementa, el valor $s_2[i]$ se comporta de la misma manera. Un valor de -1 indica una correlación total negativa, es decir, cuando un valor $s_1[i]$ incrementa o decrementa, el valor $s_2[i]$ se comporta de manera inversa. Por último, un valor de 0 indica que los valores de s_1 y s_2 no tienen correlación alguna.

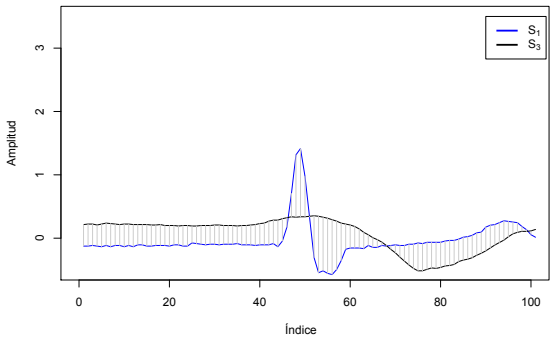
$$ICC(s_1, s_2) = \frac{covar(s_1, s_2)}{\sqrt{var(s_1)var(s_2)}} \tag{3.2}$$

$$covar(s_1, s_2) = \frac{\sum_{i=0}^{n-1}(s_1[i] - \mu_{s_1})(s_2[i] - \mu_{s_2})}{n} \tag{3.3}$$

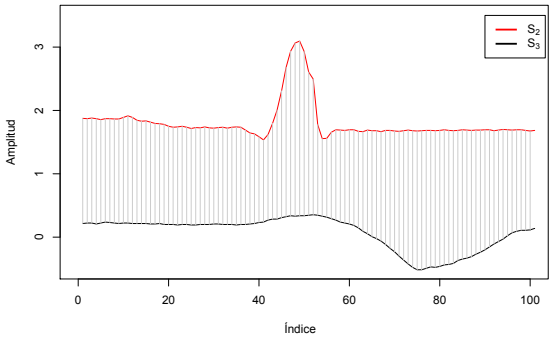
$$var(s) = \frac{\sum_{i=0}^{n-1}(s[i] - \mu_s)^2}{n} \tag{3.4}$$



(a) Semejanza Punto a Punto Entre las Secuencias S_1 y S_2.



(b) Semejanza Punto a Punto entre las Secuencias S_1 y S_3.



(c) Semejanza Punto a Punto Entre las Secuencias S_2 y S_3.

Figura 3.4: Representación Gráfica del Cálculo de Semejanza Punto a Punto.

Dadas las secuencias S_1, S_2 y S_3 de longitud n (ilustradas en la Figura 3.3), el cálculo de semejanza utilizando ICC se realiza al igual que con distancia Euclideana, una comparación punto a punto de sus valores (Figura 3.4). El resultado al aplicar ICC sin embargo, mide la correlación que existe entre los puntos, determinando una semejanza sobre la forma más que sobre la distancia entre los puntos. Basado en ello, ICC establece, tal y como se ilustra en la Figura 3.7b, que las secuencias más semejantes son S_1 y S_2, con un valor de semejanza de 0.99, lo que representa un alto índice de correlación. Por otro lado,

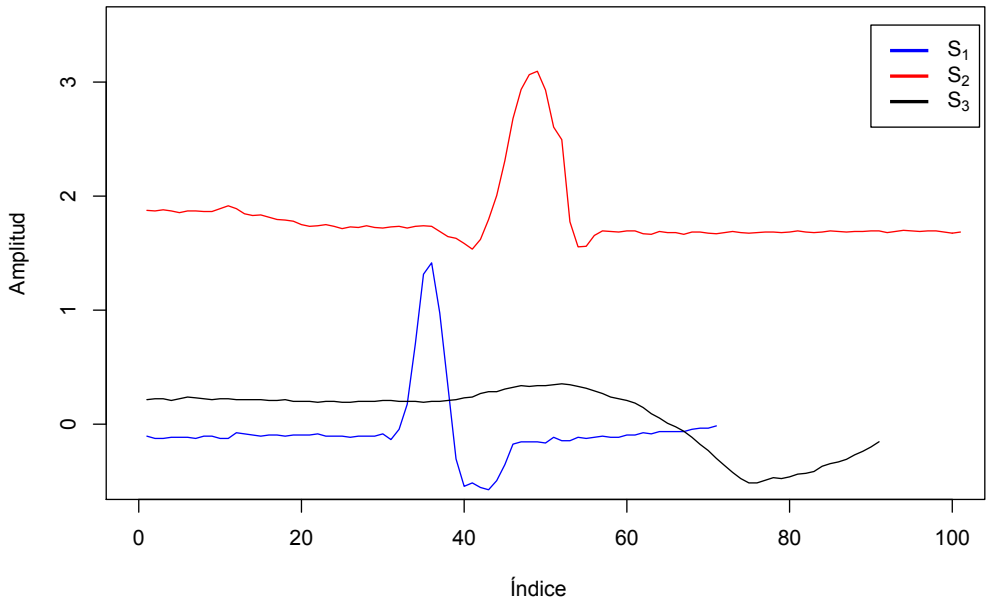
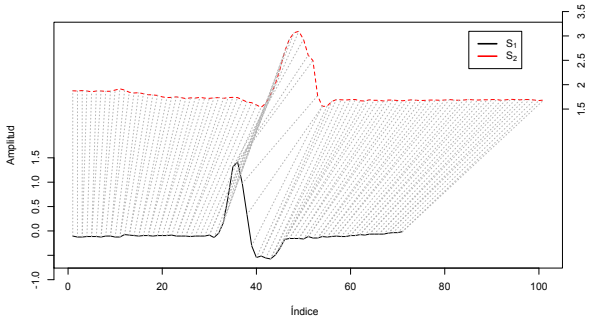


Figura 3.5: Tres Secuencias Denominadas S_1,S_2 y S_3 con Longitudes $n_1 = 70$,$n_2 = 100$ y $n_3 = 90$.

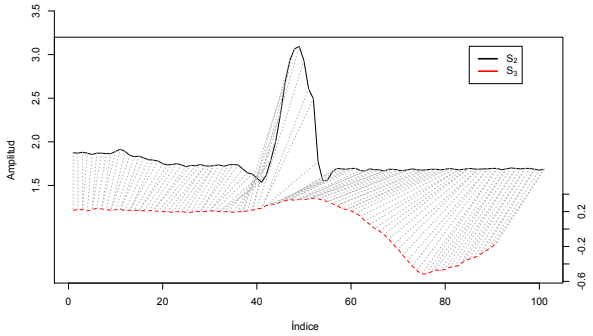
la semejanza entre S_1 y S_3 tiene un valor de 0.2, es decir, no existe un grado alto de correlación entre ellas.

- **Distorsión Dinámica de Tiempo (DTW)**: Al calcular la distancia Euclideana o índice de correlación cruzada entre dos secuencias s_1 y s_2, el i-ésimo valor de s_1 es comparado contra el i-ésimo valor de s_2. La intuición detrás de DTW es que no podemos esperar que un evento (o un patrón específico) aparezca en la misma posición o que sea de la misma longitud. Es por ello, que DTW captura la semejanza entre las formas de s_1 y s_2 de manera que el i-ésimo valor de s_1 no es necesariamente comparado con el i-ésimo valor de s_2, sino que puede ser comparado con el i'-ésimo valor de s_2.

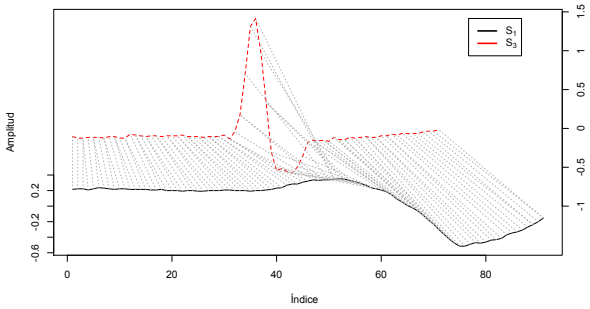
  Dadas tres secuencias S_1,S_2 y S_3 de longitud n_i (Figura 3.5), DTW determina la semejanza entre dos de ellas calculando la distancia de los puntos j y j',



(a) Cálculo de DTW Entre las Secuencias S_1 y S_2.



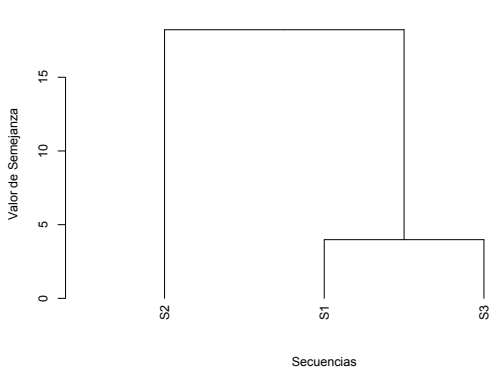
(b) Cálculo de DTW Entre las Secuencias S_1 y S_3.



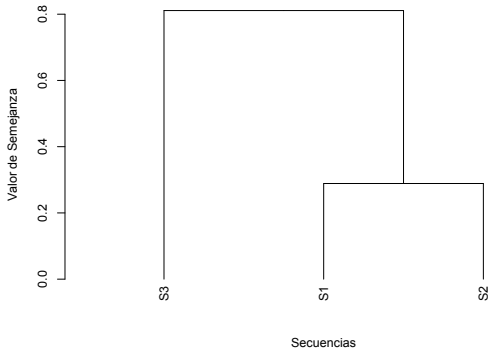
(c) Cálculo de DTW Entre las Secuencias S_2 y S_3.

Figura 3.6: Representación Gráfica del Cálculo de Semejanza Utilizando DTW.

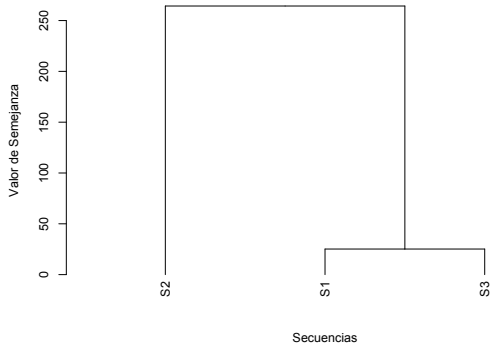
donde no necesariamente $j = j'$, como se ilustra en la Figura 3.6. El resultado de DTW establece que las secuencias S_1 y S_3 presentan un valor de semejanza mayor con respecto al obtenido entre las secuencias S_1 y S_2. Este resultado puede ser observado en la Figura 3.7c.



(a) Resultados de Distancia Euclideana.



(b) Resultados de Índice de Correlación Cruzada.



(c) Resultados de DTW.

Figura 3.7: Aglomeración de Secuencias De Acuerdo a los Distintos Algoritmos de Cálculo de Semejanza.

3.2.2. Búsqueda por Vecinos más Cercanos (K-NN)

La búsqueda de subsecuencias *top-k* se centra en la problemática K-NN, donde se requiere encontrar las k subsecuencias s'_i dentro de S consideradas como las más semejantes a q. De esta manera se obtiene un conjunto R con tripletas de la forma $(i, p, Sem(s'_i, q))$, donde $|R| = k$. Los k elementos que componen a R son conocidos como los vecinos mas cercanos a q en S. El Algoritmo 1 describe los pasos utilizados para resolver el problema K-NN en la búsqueda de subsecuencias.

Algoritmo 1 Algoritmo Para Resolver Búsqueda por Vecinos más Cercanos (K-NN)

Entrada: Relación S,
Patrón de interés q,
Valor de k,
Función de semejanza $Sem()$
Salida: Conjunto de tripletas $R = \{(i, p, Sem(s'_i, q))\}$

```
1: hacer semejanzas = {}
2: hacer R = {}
3: Para todo s_i en S Hacer
4:    hacer indice = 0
5:    Para todo s'_i en s_i Hacer
6:       calcular semejanza = Sem(s'_i, q)
7:       hacer semejanzas = semejanzas ∪ {(i, indice, semejanza)}
8:       hacer indice = indice + 1
9:    Fin Para
10: Fin Para
11: hacer semejanzas_ord = ordenar(semejanzas, semejanza)
12: hacer R = obtener_top(semejanzas_ord, k)
13: Devolver R
```

Los parámetros que recibe el Algoritmo 1 son a) un conjunto de secuencias S, b) el patrón que se quiere encontrar q, c) el número de resultados que se desean encontrar k y d) la función de semejanza $Sem(s'_i, q)$. En la Línea 3 se itera sobre todas las señales individuales s_i, y en la Línea 4 se establece el índice actual donde se buscará la primer coincidencia contra q. En la Línea 5 se generan todas las posibles subsecuencias $s'_i \in s_i$ para después calcular la semejanza contra q en la Línea 6. Todas las tripletas posibles $(i, indice, semejanza)$ son agregadas al conjunto *semejanzas* (Línea 7). Una vez agregadas todas las tripletas encontradas, se ordenan por el valor de *semejanza* (el orden dependerá directamente de la función de semejanza utilizada en el cálculo de $Sem(s'_i, q)$) en la Línea 11. Como paso siguiente, se obtienen los primeros k valores que representan las mejores semejanzas con respecto a q (Línea 12). Por último, en la Línea 13 se regresa el conjunto R que contiene la información de las k subsecuencias más semejantes a q encontradas en S.

3.2.3. Búsqueda por Rango (ϵ-search)

La búsqueda de subsecuencias por rango se centra en la problemática ϵ-search para encontrar todas aquellas subsecuencias s'_i dentro de S, tal que $Sem(s'_i, q) \leq \epsilon$. De esta manera se obtiene un conjunto R con tripletas de la forma $(i, p, Sem(s'_i, q))$. El Algoritmo 2 describe los pasos utilizados para resolver el problema ϵ-search en la búsqueda de subsecuencias.

Algoritmo 2 Algoritmo Para Resolver Búsqueda por rango (ϵ-search)

Entrada: Relación S,
Patrón de interés q,
Valor de ϵ,
Función de semejanza $Sem()$
Salida: Conjunto de tripletas $R = \{(i, p, Sem(s'_i, q))\}$

```
 1: hacer semejanzas = {}
 2: hacer R = {}
 3: Para todo s_i en S Hacer
 4:    hacer indice = 0
 5:    Para todo s'_i en s_i Hacer
 6:       calcular semejanza = Sem(s'_i, q)
 7:       Si semejanza ≤ ε Entonces
 8:          hacer R = R ∪ {(i, indice, semejanza)}
 9:       Fin Si
10:       hacer indice = indice + 1
11:    Fin Para
12: Fin Para
13: Devolver R
```

La descripción del Algoritmo 2 es la siguiente. Al igual que el algoritmo para resolver K-NN, ϵ-search recibe como parámetros a) un conjunto de secuencias S, b) el patrón q que se desea encontrar y c) la función de semejanza $Sem(s'_i, q)$. Sin embargo, en lugar de recibir un valor fijo que representa el número de resultados devueltos por el algoritmo, ϵ-search recibe un valor de umbral ϵ, de tal manera que todas las semejanzas $Sem(s'_i, q) \leq \epsilon$ se consideran como parte del conjunto de salida R. En términos generales, desde la Línea 1 hasta la Línea 6, el algoritmo ϵ-search

presenta el mismo funcionamiento que el algoritmo para resolver K-NN. En la Línea 7 el algoritmo realiza un filtrado de todos aquellos valores de $Sem(s'_i, q) \leq \epsilon$ puesto que todos ellos formarán parte del conjunto de salida R. Finalmente en la Línea 13 el algoritmo regresa el conjunto de salida R.

3.2.4. Operadores Básicos en Algoritmos de Búsqueda por Semejanza

Una vez que los algoritmos que se desean implementar en el SGBD han sido seleccionados y analizados, el siguiente paso es identificar aquellos operadores básicos que los componen y agruparlos de tal manera que cada algoritmo se pueda formar a partir de dichos operadores. Esta agrupación se realiza con el fin de no implementar cada uno de los algoritmos como una función ad-hoc sobre el SGBD. En su lugar, se propone dar al SGBD la capacidad de proveer dichos algoritmos como una serie de operadores que pueden a su vez ser reutilizados en la construcción de diversas consultas que representan operaciones utilizadas dentro del campo del procesamiento de señales de ECG digitales.

KNN Sobre el Modelo de Almacenamiento Horizontal

Es importante tener en cuenta que cada uno de los modelos de almacenamiento que se utilicen contempla una manera distinta de tratar los datos. Cuando los datos se encuentran almacenados sobre el modelo de almacenamiento horizontal, para el algoritmo que resuelve KNN se identifican los operadores ilustrados en la Figura 3.8. En ella podemos identificar cinco operadores básicos (marcados con rectángulos) que componen el algoritmo para poder resolver KNN sobre el modelo de almacenamiento horizontal. La entrada del algoritmo es un conjunto S con N secuencias, un patrón de interés q y el valor de k. El primer operador denominado $HSim$ se encarga de obtener la semejanza entre una subsecuencia $s'_i \in S$ y un patrón de interés q, para generar así un conjunto de secuencias S_d, donde cada secuencia sd_i contiene

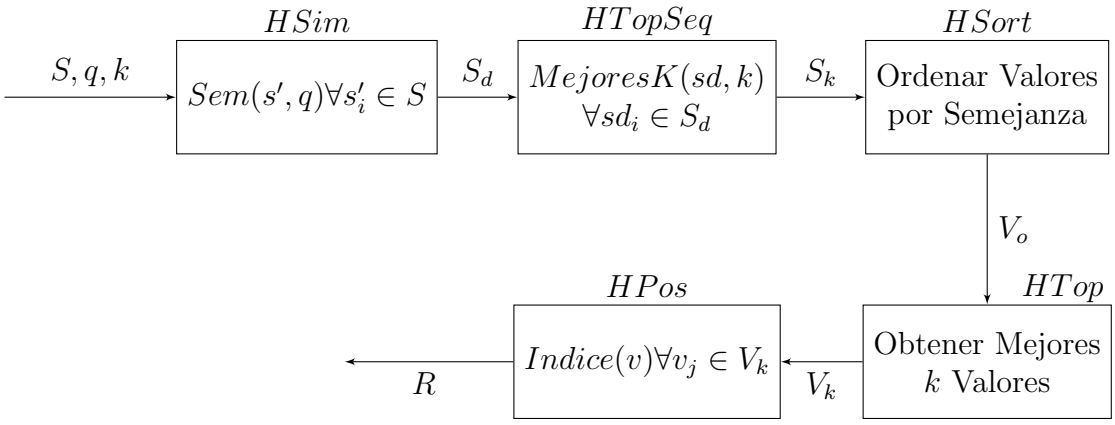


Figura 3.8: Operadores Identificados en el Cálculo de K-NN Sobre el Modelo de Almacenamiento Horizontal.

las semejanzas entre la secuencia s_i y el patrón de interés q. El segundo operador identificado como $HTopSeq$ es necesario para obtener los mejores k valores de cada una de las secuencias en S_d, generando otro conjunto S_k con secuencias de la forma $sk_i = (sk_i[0], ..., sk_i[k-1])$. Una vez generado el conjunto S_k, un tercer operador denominado $HSort$ es necesario para integrar todos los valores encontrados dentro de S_k y ordenarlos de acuerdo al valor de semejanza, generando una secuencia de valores ordenados $V_o = (vo_1, vo_2, ..., vo_{kN})$. Un cuarto operador identificado como $HTop$ se encarga de seleccionar los mejores k valores de una secuencia de valores V_o, con lo que obtenemos una secuencia $V_k = (vk_1, vk_2, ..., vk_k)$. Como operador final, $HPos$ encuentra la secuencia original a la cual pertenece cada uno de los valores de semejanza vk, así como la posición donde se encontró dicha semejanza, generando el conjunto de salida $R = \{(i, p, vk)\}$.

KNN Sobre el Modelo de Almacenamiento Vertical

Los operadores identificados para el algoritmo que resuelve KNN sobre el modelo de almacenamiento vertical se observan en la Figura 3.9. Dado un conjunto S con N secuencias, un patrón de interés q y un valor de k, el primer operador denominado $VSim$ es necesario para el cálculo de semejanza entre una subsecuencia $s'_i \in S$ y q, lo

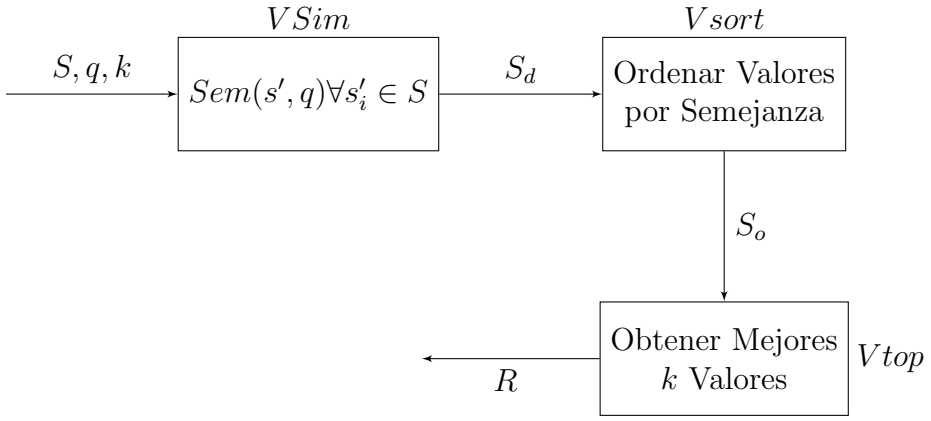


Figura 3.9: Operadores Identificados en el Cálculo de K-NN Sobre el Modelo de Almacenamiento Vertical.

que genera un conjunto de tripletas de la forma $S_d = \{(i, p, Sem(s', q))\}$, donde i es el índice que identifica la secuencia original, p es la posición donde comienza la subsecuencia, y $Sem(s', q)$ es el valor de semejanza entre la subsecuencia s' y q. El segundo operador llamado $Vsort$ se encarga de ordenar los valores de S_d de acuerdo al valor de semejanza, lo que genera una serie de tripletas ordenadas $S_o = ((i, p, Sem(s', q)))$. Finalmente, un tercer operador denominado $VTop$ obtiene directamente las mejores k tripletas de S_o y genera así el conjunto de salida R. Es importante mencionar que a pesar de que los operadores tienen la misma función que los operadores encontrados en el almacenamiento horizontal estos no pueden ser reutilizados debido a la forma en que los datos se encuentran almacenados, y por tanto, sus parámetros de entrada y salida son diferentes.

ϵ-search Sobre el Modelo de Almacenamiento Horizontal

Algunos de los operadores identificados dentro del algoritmo para resolver ϵ-search en el modelo de almacenamiento horizontal pueden ser reutilizados desde algoritmo para K-NN. En la Figura 3.10 se observan en primer lugar, los datos de entrada: un conjunto S con N secuencias, un patrón de interés q y el valor de umbral ϵ. El primer operador $HSim$ puede ser reutilizado para para obtener la semejanza entre

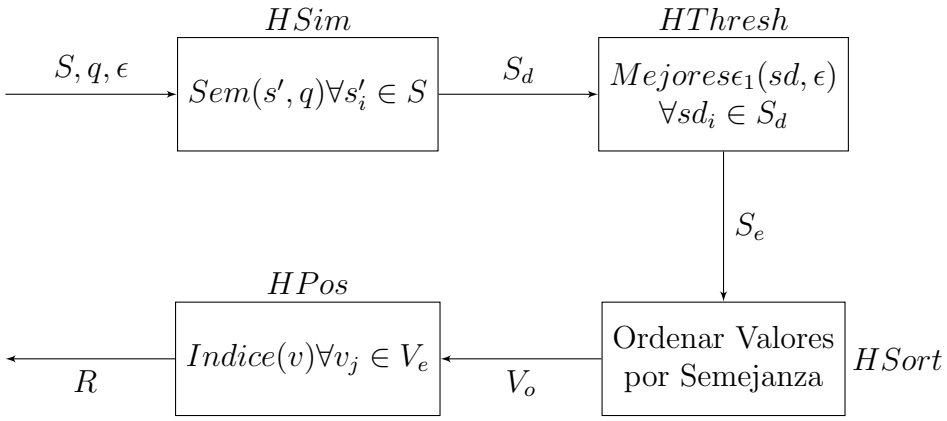


Figura 3.10: Operadores Identificados en el Cálculo de ϵ-search Sobre el Modelo de Almacenamiento Horizontal.

una subsecuencia $s'_i \in S$ y un patrón de interés q. El segundo operador denominado $HThresh$ es necesario para obtener aquellos valores de semejanza por encima del umbral ϵ en cada una de las secuencias encontradas en S_d, generando otro conjunto S_e con secuencias de la forma $se_i = (se_i[0], ..., se_i[j-1])$. Una vez generado el conjunto S_e, el tercer operador $HSort$ es reutilizado para integrar todos los valores encontrados dentro de S_e y ordenarlos de acuerdo al valor de semejanza, generando una secuencia de valores ordenados $V_o = (vo_1, vo_2, ..., vo_j)$. Por último, el cuarto operador $HPos$ también puede ser reutilizado para encontrar cual es la secuencia original a la cual pertenece cada uno de los valores de semejanza vo, así como la posición donde se encontró dicha semejanza, generando el conjunto de salida $R = \{(i, p, vo)\}$.

ϵ-search Sobre el Modelo de Almacenamiento Vertical

Los operadores identificados para el algoritmo que resuelve ϵ-search sobre el modelo de almacenamiento vertical se ilustran en la Figura 3.11. Dado un conjunto S con N secuencias, un patrón de interés q y el valor de umbral de semejanza ϵ, el primer operador $VSim$ es reutilizado del algoritmo K-NN sobre el modelo de almacenamiento vertical para el cálculo de semejanza entre una subsecuencia $s'_i \in S$ y q. El segundo operador $VSort$ también es reutilizado del algoritmo K-NN para ordenar los

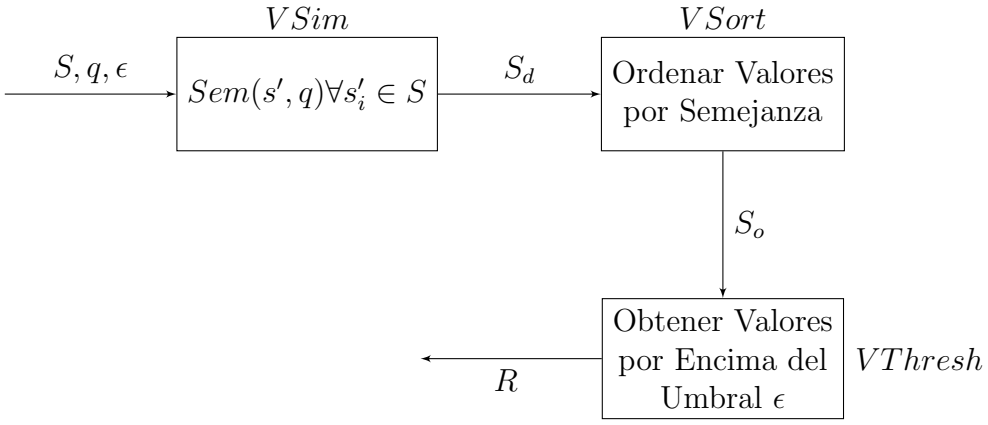


Figura 3.11: Operadores Identificados en el Cálculo de ϵ-search Sobre el Modelo de Almacenamiento Vertical.

valores de S_d de acuerdo al valor de semejanza, lo que genera una serie de tripletas $S_o = ((i, p, Sem(s', q)))$. Finalmente con un tercer operador denominado $VThresh$ podemos obtener directamente todas las tripletas que componen el conjunto de salida R donde el valor de $Sem(s', q)$ se encuentre por encima del umbral de semejanza ϵ.

3.3. Extensión de un SGBD Utilizando Funciones Definidas por el Usuario

Posteriormente a la selección de algoritmos que se desean ser agregados el SGBD, y los operadores necesarios han sido identificados es necesaria la implementación de funciones para que los operadores sean utilizados correctamente. Cuando la integración de cálculos dentro de un SGBD no cuenta con las funciones apropiadas o suficientes para su realización, se vuelve necesaria la extensión de la funcionalidad del SGBD. Esta facilidad es ofrecida por algunos SGBD comerciales actuales, tales como PostgreSQL, MySQL, Teradata y SQL-Server (entre otros), con el uso de funciones definidas por el usuario (UDFs, *User Defined Functions*). Una UDF forma parte del estándar SQL, y se define como una subrutina escrita en un lenguaje de programa-

ción de alto nivel (C o C++, por ejemplo) [49]. Una vez escrita, la UDF es compilada y después enlazada con el propio SGBD. De esta manera, la UDF puede ser llamada mediante la utilización de una consulta de tipo SELECT. Ordoñez y García [49] han observado que, dado que las UDF pueden ser escritas utilizando el lenguaje C, es posible también utilizar las estructuras de control que el propio lenguaje brinda, por ejemplo "if", "while", "for", para poder expresarlas dentro de la UDF. De esta manera también es posible utilizar arreglos multidimensionales dentro de una UDF (aunque en algunos SGBD comerciales actuales no es un tipo de dato que pueda ser proporcionado a través de una consulta SELECT).

Tipos de UDF

Cada uno de los SGBD comerciales actuales contiene un conjunto bien definido de funciones incorporadas. En los trabajos de Chen [9] y, Jaedicke y Mitschang [27], se identifican los siguientes tipos de UDF en los SGBD: escalares, de agregación y de tablas.

- UDF Escalar (UDFE): Una UDFE es aplicada sobre el valor de varios atributos de una misma tupla que recibe como entrada. Dicho de otra manera, por cada una de las tuplas recibida como parámetro, una UDFE regresa un solo valor escalar como resultado. Ejemplos de funciones escalares en el estándar SQL son las funciones aritméticas suma(+) y multiplicación ($\times$).

- UDF de Agregación (UDFA): Una UDFA es aplicada sobre los valores de un mismo atributo, ya sea de un grupo de tuplas o considerando a toda la relación como un solo grupo. La salida de una UDFA es un valor escalar por cada grupo de tuplas, el cual es definido cuando se utiliza la cláusula de SQL: GROUP BY. Ejemplos de funciones de agregación en el estándar SQL son las funciones AVG() y COUNT().

- UDF de ventana: Una UDF de ventana realiza un cálculo sobre conjuntos de tuplas en una tabla que de alguna manera se encuentran relacionadas hacia la tupla actual. Este tipo de función puede compararse con una UDFA. Sin embargo, a diferencia de las UDFA, las UDF de ventana no generan una sola tupla al procesar un conjunto de tuplas. En su lugar, cada tupla mantiene su resultado de manera independiente. Una UDF de ventana siempre contiene una cláusula OVER posterior al nombre de la función en sí seguida de los argumentos de la función. Esto último es lo que distingue a las funciones de ventana de las funciones escalares o de agregación. La cláusula PARTITION BY determina como las tuplas se van a agrupar o particionar para ser procesadas por la función de ventana. El alcance de la ventana puede ser establecido con la cáusula ROWS BETWEEN que determina los límites a partir de la tupla actual que se esté procesando. Para cada tupla dentro de una partición la función de ventana calcula el valor incluyendo las demás tuplas que se encuentran dentro de dicha partición. También se puede utilizar la cláusula ORDER BY para determinar el orden de las tuplas dentro de una partición. Ejemplos de funciones de ventana en el estándar SQL son las funciones RANK() y ROW_NUM(). Cabe mencionar que cualquier función de agregación puede ser utilizada como función de ventana.

- UDF de tablas (UDFT): Una UDFT, a diferencia de una UDFE y una UDFA, puede regresar un conjunto de tuplas en lugar de un solo valor. Una UDFT sólo puede tener argumentos escalares, desde uno hasta n, y retorna un conjunto de tuplas. Cada una de estas tuplas en el conjunto de salida puede consistir en uno o múltiples atributos.

3.4. Propuesta General de Consultas Necesarias

Con la finalidad de poder implementar búsqueda por semejanza dentro de un SGBD, es necesario proponer consultas que nos permitan utilizar los algoritmos que resuelven KNN y ϵ-search sobre las señales almacenadas tanto en el modelo de almacenamiento horizontal como en el modelo de almacenamiento vertical. En esta sección se propone la estructura general que las consultas a desarrollar sobre el estándar SQL deben cumplir. En ellas se implementan los operadores básicos identificados en la Sección 3.2.4 como una serie de consultas que nos permiten obtener los resultados deseados por ambos algoritmos. De igual manera se establecen las funciones que se necesitan para implementar tanto K-NN como ϵ-search como funciones nativas o como UDFs. Dada una relación que contiene secuencias $S = \{s_1, s_2, ...s_N\}$ y un patrón de interés q, a continuación se describe la estructura general propuesta para el desarrollo de consultas que resuelven K-NN y ϵ-search en un SGBD.

3.4.1. K-NN Sobre el Modelo de Almacenamiento Horizontal

La estructura general para las consultas que resuelvan K-NN sobre el modelo de almacenamiento horizontal deberán de presentar una serie de subconsultas que contemplen los operadores detectados en la Sección 3.2.4 e ilustrados en la Figura 3.8. De esta manera, para implementar K-NN sobre el modelo de almacenamiento horizontal se requiere de las siguientes consultas generales.

- La primera consulta representa el operador $HSim$. Se encarga de obtener una relación S_d que contiene N secuencias sd_i y sus respectivos identificadores. Cada una de las secuencias sd_i contiene todos aquellos valores de semejanza que existen entre cada una de las posibles subsecuencias s'_i y un patrón de interés q utilizando una UDF para el cálculo de semejanza. A esta UDF la que denominamos $Sem(s_i, q)$. La consulta expresada en SQL tiene la siguiente forma.

  SELECT i, Sem(s_i,q)
  INTO S_d
  FROM S;

  Donde i es identificador de la secuencia s_i y $Sem(s_i, q)$ calcula la semejanza entre todas las posibles subsecuencias s'_i y un patrón q (con $|s'_i| = |q|$).

- La segunda consulta representa el operador $HTopSeq$. Se utiliza para generar una relación S_k, que almacena N secuencias sk_i y sus respectivos identificadores. Cada una de las secuencias sk_i contiene los mejores k valores encontrados en $sd_i \in S_d$. Es necesario utilizar una UDF para obtener S_k. A esta UDF la denominamos $KminValues(sd_i, k)$, donde k es el número de mejores valores de semejanza que se desea obtener de sd_i. La consulta expresada en SQL tiene la siguiente forma.

  SELECT i, KminValues(sd_i,k)
  INTO S_k
  FROM S_d;

- La tercera consulta representa el operador $HSort$ y genera una relación V_o que contiene todos los valores ordenados de S_k, así como el identificador i de la secuencia sk_i a la que pertenecen. La consulta necesaria, expresada en SQL tiene la siguiente forma.

SELECT i, unnest(sk_i) as value
INTO V_o
FROM S_k
ORDER BY values ASC;

Donde i es identificador de la secuencia sk_i y $unnest(sk_i)$ es una función de SQL definida en el estándar SQL:1999, la cual descompone los valores de una secuencia en un conjunto de tuplas con los valores en el orden en que se encontraban dentro de la secuencia.

- Una cuarta consulta representa el operador $HTop$. Esta consulta se utiliza para obtener una relación V_k que contiene un conjunto de k valores y el idenficador de la secuencia a la cual pertenecen. Estos valores representan los k mejores valores encontrados en V_o. La consulta expresada en SQL tiene la siguiente forma.

  SELECT i, value
  INTO V_k
  FROM V_o
  LIMIT k;

- Finalmente, una quinta consulta que representa el operador $HPos$ es necesaria poder encontrar la posición de cada uno de los valores de V_k en la señal s_i donde fué calculado. Para ello proponemos una UDF a la cuál denominamos $Pos(value, sd_i)$. De esta manera, podemos obtener una relación R que contiene k tripletas de la forma $(i, p, Sem(s'_i, q))$. Esta relación representa la salida a la problemática $K-NN$. La consulta expresada en SQL tiene la siguiente forma.

  SELECT i, $Pos(value, sd_i)$ as p, value
  INTO R
  FROM V_k, S_d
  WHERE V_k.i = S_d.i;

3.4.2. K-NN Sobre el Modelo de Almacenamiento Vertical

Para aplicar K-NN sobre el modelo de almacenamiento vertical es necesario que se utilize una serie de consultas que contemplen los operadores detectados en la Sección 3.2.4 e ilustrados en la Figura 3.9. A continuación se presenta la estructura general que las consultas deben seguir para poder implementar de manera adecuada K-NN utilizando el modelo de almacenamiento vertical.

- La primera consulta representa el operador $VSim$ y se encarga de obtener una relación S_d que contiene tripletas de la forma $(i, t, v_Sem(s'_i, q))$ para cada secuencia presente en S, donde i es el identificador de la secuencia s_i y t es la posición en s_i donde comienza s'_i. Para ello es necesario implementar la UDF $v_Sem(s_i, q)$ como una función de ventana, descrita en la Sección 3.3. La consulta expresada en SQL tiene la siguiente forma.

  SELECT i, t, v_Sem(s_i,q) as sem
  OVER (PARTITION BY i ORDER BY t ASC
  ROWS BETWEEN CURRENT ROW AND n FOLLOWING)
  INTO S_d
  FROM S;

  Donde la UDF $v_Sem(s_i, q)$ calcula la semejanza entre todas las posibles subsecuencias de s_i y un patrón q (con $n = |q|$).

- Una segunda consulta representa el operador $VSort$ la cual ordena los valores de semejanza generados por la consulta anterior. El resultado de esta consulta es una relación S_o con tripletas de la forma (i, t, sem), ordenadas por el valor de semejanza sem. De esta manera, la consulta necesaria tiene la siguiente forma.

  SELECT i, t, sem
  INTO S_o
  FROM S_d
  ORDER BY sem ASC;

- Una última consulta representa el operador $VPos$. Esta consulta obtiene los mejores k valores que se encuentren en S_o, con lo que se obtiene una relación de salida R que contiene k tripletas de la forma $(i, p, Sem(s'_i, q))$, la cual representa la salida en la problemática K-NN.

  SELECT i, t, sem
  INTO R
  FROM S_o
  LIMIT k;

3.4.3. ϵ-search Sobre el Modelo de Almacenamiento Horizontal

Para resolver la problemática ϵ-search sobre el modelo de almacenamiento horizontal se propone la siguiente serie de consultas que contemplen los operadores detectados en la Sección 3.2.4 e ilustrados en la Figura 3.10.

- La primera consulta representa al operador $HSim$ y es reutilizada del algoritmo KNN. Se encarga de obtener una relación S_d la cual contiene N secuencias sd_i y sus respectivos identificadores. Cada secuencia sd_i contiene todos aquellos valores de semejanza que existen entre cada una de las posibles subsecuencias s'_i y un patrón de interés q utilizando una UDF para el cálculo de semejanza, a la cual denominamos $Sem(s_i, q)$. La consulta expresada en SQL tiene la siguiente forma.

  SELECT i, Sem(s_i,q)
  INTO S_d
  FROM S;

  Donde i es identificador de la secuencia s_i y $Sem(s_i, q)$ es una UDF que calcula la semejanza entre todas las posibles subsecuencias s'_i y un patrón q (con $|s'_i| = |q|$).

- La segunda consulta representa el operador $HThresh$ y genera una relación S_ϵ, que almacena N secuencias $s\epsilon_i$ y sus respectivos identificadores. Cada una

de las secuencias $s\epsilon_i$ contiene todos aquellos valores superiores al umbral ϵ_1 encontrados en $sd_i \in S_d$ utilizando una UDF. A esta UDF la denominamos $EminValues(sd_i, \epsilon)$, donde ϵ es un umbral de semejanza establecido. La consulta expresada en SQL tiene la siguiente forma.

```
SELECT i, EminValues(sd_i,ε)
INTO S_ε
FROM S_d;
```

- La tercera consulta representa el operador $Hsort$ reutilizado del algoritmo K-NN. Esta consulta genera una relación V_o que contiene todos los valores ordenados de S_ϵ, así como el identificador i de la secuencia $s\epsilon_i$ a la que pertenecen. La consulta necesaria, expresada en SQL tiene la siguiente forma.

```
SELECT i, unnest(sε_i) as value
INTO V_o
FROM S_ε
ORDER BY values ASC;
```

Donde i es identificador de la secuencia $s\epsilon_i$ y $unnest(s\epsilon_i)$ es una función estándar que descompone los valores de una secuencia en un conjunto de tuplas con los valores en orden el que se encontraban dentro de la secuencia.

- Una última consulta representa el operador $HPos$, también reutilizada del algoritmo K-NN. Esta consulta encuentra la posición de cada uno de los valores en V_o utilizando una UDF denominada $Pos(value, sd_i)$, la cual busca la posición donde se encuentra $value$ dentro de sd_i. De esta manera, podemos obtener una relación R que contiene j tripletas de la forma $(i, p, Sem(s'_i, q))$, la cuál representa la salida en la problemática ϵ-search. La consulta expresada en SQL tiene la siguiente forma.

SELECT i, $Pos(value, sd_i)$ as p, value
INTO R
FROM V_k, S_d
WHERE V_k.i = S_d.i;

3.4.4. ϵ-search Sobre el Modelo de Almacenamiento Vertical

Por último, para implementar ϵ-search sobre el modelo de almacenamiento vertical se propone el uso de la siguiente serie de consultas que contempla los operadores detectados en la Sección 3.2.4 e ilustrados en la Figura 3.11.

- La primera consulta representa el operador $VSim$ reutilizado del algoritmo KNN. Esta consulta genera una relación S_d que contiene tripletas de la forma $(i, t, v_Sem(s'_i, q))$ para cada secuencia presente en S, donde i es el identificador de la secuencia s_i y t es la posición en s_i donde comienza s'_i. La consulta expresada en SQL tiene la siguiente forma.

  SELECT i, t, v_Sem(s_i,q) as sem
  OVER (PARTITION BY i ORDER BY t ASC
  ROWS BETWEEN CURRENT ROW AND n FOLLOWING)
  INTO S_d
  FROM S;

  Donde $v_Sem(s_i, q)$ es una UDF de ventana que calcula la semejanza entre todas las posibles subsecuencias de longitud n en s_i y un patrón q (con $n = |q|$).

- La segunda consulta representa el operador $VSort$ reutilizado también del algoritmo KNN. Esta consulta se utiliza para ordenar los valores de semejanza generados por la consulta anterior. El resultado de esta consulta es una relación S_o con tripletas de la forma (i, t, sem), ordenadas por el valor de semejanza sem. De esta manera, la consulta necesaria tiene la siguiente forma.

```
SELECT i, t, sem
INTO S_o
FROM S_d
ORDER BY sem ASC;
```

- Finalmente, una tercera consulta representa el operador $VThresh$ y conserva sólo aquellos valores en S_o que se encuentren por encima del umbral de semejanza ϵ. Con ello se genera una relación R que contiene j tripletas de la forma $(i, p, Sem(s'_i, q))$, la cual representa la salida en la problemática ϵ-search. La consulta expresada en SQL tiene la siguiente forma.

```
SELECT i, t, sem
INTO R
FROM S_o
WHERE sem >= ε
ORDER BY sem ASC;
```

En este Capítulo se han descrito las bases para implementar la búsqueda de subsecuencias sobre señales de ECG digitales dentro de un SGBD. Hemos propuesto la estructura de dos modelos distintos para el almacenamiento de las señales de ECG digitales utilizando los tipos de datos nativos de un SGBD para representar sus atributos. Para cada uno de los modelos de almacenamiento proponemos también la estructura general de las consultas necesarias para poder realizar las operaciones de búsqueda por semejanza bajo las problemáticas K-NN y ϵ-search. Las consultas propuestas en este trabajo de tesis están compuestas a su vez por una serie de subconsultas. Cada una de las subconsultas representa un operador identificado en la Sección 3.2.4 para los algoritmos de búsqueda por semejanza. En la Tabla 3.1 se observa un resumen del núcleo de operadores identificados así como su reutilización entre los dos algoritmos de búsqueda por semejanza y los modelos de almacenamiento propuestos.

Con la propuesta de consultas compuestas por operadores y la reutilización de estos últimos evitamos la implementación de cada uno de los algoritmos como una

Operador	Algoritmo+Modelo de Almacenamiento			
	K-NN+Hor	K-NN+Ver	ϵ-search+Hor	ϵ-search+Ver
HSim	•		•	
HTopSeq	•			
HSort	•		•	
HTop	•			
HPos	•		•	
HThresh			•	
VSim		•		•
VSort		•		•
VTop		•		
VThresh				•

Tabla 3.1: Reutilización de Operadores Identificados Entre Algoritmos y Modelos de Almacenamiento

función ad-hoc del SGBD. Para la creación de las consultas se contempló un conjunto de funciones nativas del SGBD. Sin embargo, ante la carencia de algunas funciones requeridas para lograr la implementación de búsqueda por semejanza se aprovecha la facilidad que ofrecen algunos SGBD comerciales actuales para incrementar sus capacidades utilizando UDFs. De esta manera queda establecida la metodología completa para implementar la búsqueda por semejanza haciendo uso exclusivo de un SGBD aprovechando las capacidades que este nos brinda.

Capítulo 4
Evaluación Experimental y Resultados

La experimentación realizada en este trabajo de tesis contempla la medición de eficiencia para las consultas propuestas durante la implementación de los algoritmos de búsqueda por semejanza sobre señales de ECG digitales dentro de un SGBD. La eficiencia fue medida en términos del tiempo de ejecución requerido por cada una de las consultas utilizadas. Las consultas propuestas resuelven los algoritmos de vecinos más cercanos (K-NN, *K-Nearest Neighbours*) y ϵ-search en los modelos de almacenamiento horizontal y almacenamiento vertical.

La plataforma computacional en la que fueron ejecutadas y evaluadas las consultas utilizadas consiste en un equipo de cómputo estacionario con un procesador Intel dual-core de 2.3Ghz, 2GB de memoria principal y un disco duro SATA III de 3.5", 7200 revoluciones por minuto (rpm) y 1 Terabyte de capacidad de almacenamiento. El SGBD seleccionado fue PostgreSQL en su versión 9.3 sobre el sistema operativo CentOS 6.5.

El tiempo de ejecución fue considerado como el tiempo requerido para obtener el conjunto de tripletas $R = \{(i, indice, Sem(s'_i, q))\}$ dado un conjunto de señales de ECG digitales S y un patrón de interés q. La cardinalidad de R, denotada como $|R|$ es un valor que depende directamente del valor asignado a k en el caso de K-NN. Para el caso de búsqueda por rango o ϵ-search el resultado depende directamente del valor asignado a ϵ. La medida de semejanza utilizada para el cálculo de $Sem(s'_i, q)$ fue la distancia Euclideana debido a su vasta utilización en la literatura. Las consultas utilizadas en nuestra experimentación fueron desarrolladas en el lenguaje SQL, bajo el estándar SQL:1999. Cada una de las consultas fue tratada para obtener el mismo resultado ante los modelos de almacenamiento horizontal y almacenamiento vertical.

El conjunto de datos de prueba utilizado proviene del repositorio Physionet, y se describe en la Sección 4.1.

En este Capítulo se contemplan distintas alternativas para el manejo intermedio de datos generados entre subconsultas. Esto último partiendo de la idea de que los tiempos de ejecución de las consultas difieren de manera significativa tanto por el modelo de almacenamiento utilizado como por el tamaño del conjunto de pruebas. Para seleccionar las alternativas utilizadas fue necesario considerar las características y herramientas especiales que nos brinda el SGBD PostgreSQL. Esto con la finalidad de explorar nuestras opciones y poder encontrar el modelo de almacenamiento que nos permita ejecutar las consultas y obtener los resultados deseados en el menor tiempo posible.

Una propuesta que resulte competitiva a una aplicación ad-hoc desarrollada y ejecutada directamente por el sistema operativo representa uno de los mayores retos [39]. Para poder comparar el desempeño de nuestra propuesta, se desarrolló una aplicación ad-hoc utilizando el lenguaje de programación ANSI C por cada algoritmo implementado dentro del SGBD. Es decir, se desarrollaron dos aplicaciones ad-hoc para realizar búsqueda por semejanza sobre los datos almacenados en formato binario como archivos del sistema operativo. Con ello, una de las aplicaciones ad-hoc y a la cual denominamos K_ADHOC se desarrolló para poder realizar búsqueda mediante el algoritmo K-NN. Por otro lado, la otra aplicación ad-hoc y a la cual denominamos E_ADHOC fue desarrollada para realizar búsqueda por semejanza con el algoritmo ϵ-search.

4.1. Descripción del Conjunto de Datos de Prueba

Los datos de prueba utilizados en este trabajo fueron obtenidos del repositorio de datos *Physionet*, accesible desde el sitio *http://physionet.org/*. Este repositorio es un recurso para aquellos investigadores que estudian señales biomédicas y series de tiempo. Physionet nos ofrece distintos tipos de señales biomédicas para el análisis de problemas abiertos a investigadores y estudiantes alrededor del mundo. Los datos disponibles en Physionet fueron diseñados de manera especial para la evaluación y aceptación de algoritmos en el análisis de señales [43].

Actualmente, en el repositorio de Physionet se cuenta con 29 bases de datos que contienen señales de ECG digitales. Estas señales se encuentran almacenadas como archivos binarios en un formato propietario con extensión ".dat" y además se proveen también archivos cabecera con extensión ".head", los cuales contienen información adicional de la señal. Nuestro trabajo contempla el uso de cuatro bases de datos del repositorio Physionet, tanto por su utilización en la literatura como por su variabilidad en duración y frecuencias de muestreo. Dicha variación nos permite obtener resultados en la aplicación de los algoritmos bajo distintos escenarios en un ambiente real. La duración y la frecuencia de las señalesde ECG digitales encontradas en el repositorio suelen ser iguales dentro de una misma base de datos, pero varían entre las distintas bases de datos [36]. La longitud de las señales de ECG digitales de las bases de datos utilizadas en nuestro trabajo varían desde los 47.93 segundos hasta las 2 horas. Por otra parte, la frecuencia de muestreo varía desde los 250Hz hasta los 720Hz [21, 39, 42]. A continuación se describen de manera mas detallada las bases de datos utilizadas en este trabajo como conjunto de datos de prueba.

- **ANSI/AAMI EC13:** Introducida al repositorio Physionet en el año 2002, es la base de datos con ECG de menor longitud. Los archivos en esta base de datos pueden ser usados para probar dispositivos en el monitoreo de ECGs online debido a su duración. Contiene 10 archivos, cada uno representa una señal de

ECG digital con una sola derivación, muestreado a 720Hz y con una resolución de 12 bits [17]. La duración de las 10 señales varían desde los 47.93 segundos hasta los 90.6 segundos [39]. En la Figura 4.1 se ilustra una subsecuencia de 2,160 valores, equivalente a 3 segundos de duración. La subsecuencia pertenece a una de las señales de ECG digitales etiquetada como 'aami3a'. A lo largo del resto de este trabajo, nos referiremos a esta base de datos como **AAMI**.

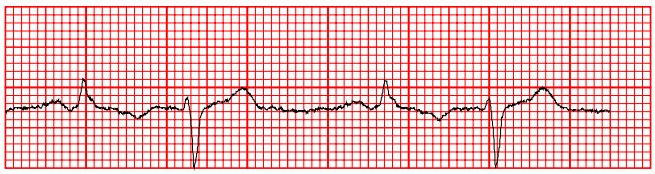


Figura 4.1: Subsecuencia de la Señal de ECG Digital de la Base de Datos ANSI/AAMI EC13 Identificada como 'aami3a'.

- **QT Database:** Esta base de datos fue agregada a Physionet en el año 2001. Contiene señales de ECG digitales que fueron seleccionados para la representación de una amplia variedad de morfologías QRS y ST-T, con la finalidad de representar un reto para los algoritmos de detección del complejo QRS en aplicaciones del mundo real. La base de datos QT contiene un total de 105 señales de ECG digitales. Cada señal de ECG digital cuenta con 2 derivaciones de 15 minutos de duración, una frecuencia de muestreo de 250Hz y una resolución de 11 bits. Cada señal de ECG digital contiene entre 30 y 100 latidos representativos anotados manualmente por cardiólogos. Estas anotaciones identifican el inicio y fin de la onda P, el inicio y fin del complejo QRS, inicio y fin de la onda T, y (de estar presente) el inicio y fin de la onda U [34]. La base de datos QT fue conformada por señales de ECG digitales tomadas de otras bases de datos (véase la Tabla 4.1). Las bases de datos que la componen son MIT-BIH

Arrhythmia [41], MIT-BIH ST [6], MIT-BIH Supraventricular [23], MIT-BIH Long Term [21], European ST database [60], MIT-BIH Normal Sinus Rithm [21] y Sudden Cardiac Death Holter DB [22]. En la Figura 4.2 se ilustra una subsecuencia de 750 valores equivalente a 3 segundos de duración. Dicha subsecuencia pertenece a la señal de ECG digital denominada 'sel100'. En el resto del presente trabajo nos referiremos a esta base de datos como **QT**.

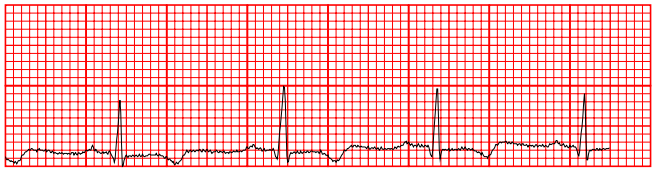


Figura 4.2: Subsecuencia de la Señal de ECG Digital de la Base de Datos QT Identificada como 'sel100'.

MIT-BIH Arrhythmia	MIT-BIH ST	MIT-BIH Supra.	MIT-BIH Long Term	Euro. ST	MIT-BIH NSR DB	Sudden Death
15	6	13	4	33	10	24

Tabla 4.1: Procedencia de las 105 Señales que Componen la Base de Datos QT.

- **MIT-BIH Arrhythmia Database:** Esta base de datos se incluyó a Physionet en el año de 1999. Fue uno de los mayores productos del esfuerzo entre el *Beth Israel Deaconess Medical Center* (BIDMC) y el *Massachusetts Institute of Technology* (MIT) tras el apoyo a la investigación hacia el análisis de arritmias y temas relacionados. Fue la primera base de datos disponible para la evaluación de detectores de arritmias. Contiene 48 señales de ECG digitales. Cada una de estas señales cuenta con 2 derivaciones obtenidas de 47 sujetos de estudio en el laboratorio de BIH Arrhythmia entre los años 1975 y 1979. Dos o más cardiólogos anotaron independientemente cada derivación. Las señales de ECG digitales

tienen una duración de 30 minutos, una frecuencia de muestreo de 360Hz y una resolución de 11 bits [41]. En la Figura 4.3 se ilustra una subsecuencia de 1080 valores equivalente a 3 segundos de duración. La subsecuencia pertenece a la señal de ECG digital etiquetada como '100'. A lo largo del resto de este trabajo, nos referiremos a esta base de datos como **MIT-BIH**.

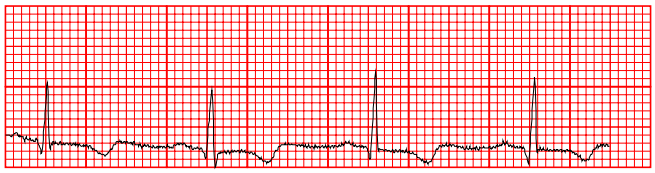


Figura 4.3: Subsecuencia de la Señal de ECG Digital de la Base de Datos MIT-BIH Arrhythmia Identificada como '100'.

- **European ST-T Database:** Esta base de datos se incluyó a Physionet en el año 2000. Fue creada con la intención de evaluar algoritmos para el análisis de los cambios en las ondas ST y T. Contiene 90 señales de ECG digitales obtenidas de 79 pacientes. Esta base de datos contiene las señales de ECG digitales con mayor longitud. Cada señal cuenta con 2 derivaciones, una duración de 2 horas, una frecuencia de muestreo de 250Hz y una resolución de 12 bits. Al igual que en la base de datos MIT-BIH Arrhythmia, 2 cardiólogos trabajaron independientemente en las anotaciones de cada derivación latido por latido. En la Figura 4.4 se ilustra una subsecuencia de 750 valores equivalentes a 3 segundos de duración, que pertenece a la señal de ECG digital etiquetada como 'e0103'. A lo largo del resto de este trabajo, nos referiremos a esta base de datos como **European**.

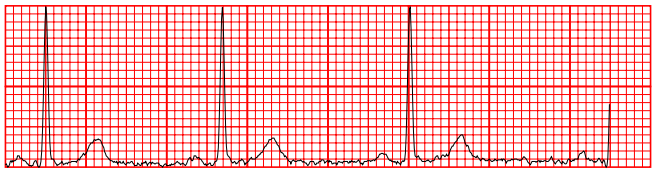


Figura 4.4: Subsecuencia de la Señal de ECG Digital de la Base de Datos European ST-T Identificada como 'e0103'.

4.2. Modelos de Almacenamiento en el Sistema Gestor de Bases de Datos

Una vez que los modelos de almacenamiento horizontal y vertical fueron definidos, lo siguiente fue realizar la implementación de estos modelos de almacenamiento dentro del SGBD. Esto último con la finalidad de poder almacenar las señales de ECG dentro del SGBD de acuerdo a los modelos de almacenamiento propuestos. Apegándonos al estándar SQL:1999, la creación de las tablas necesarias para el almacenamiento de los datos se realizó de acuerdo a los esquemas establecidos en las Secciones 3.1.1 y 3.1.2 para el almacenamiento horizontal y almacenamiento vertical, respectivamente. Para ello, cada una de las derivaciones encontradas en el conjunto de datos fue tratada como una señal de ECG digital independiente al momento de almacenarlas dentro del SGBD.

Para poder obtener los datos que se almacenaron en el SGBD fue necesaria una etapa de preprocesamiento de los datos utilizando un paquete de software denominado WaveForm DataBase Software Package (WFDB). WFDB está disponible en *Physionet* y cuenta con una aplicación llamada *rdsamp* la cuál nos permitió extraer los valores a partir de un archivo de señal de cada uno de los conjuntos de datos utilizados.

Para el caso de las aplicaciones K_ADHOC y E_ADHOC se realizó una transformación de los archivos de texto que contenían los valores de las señales de ECG

digitales a un formato binario. El resultado de esto es un archivo binario por cada una de las señales de ECG, donde cada muestra presente en una señal es un valor numérico de tipo flotante que ocupa un espacio de 4 bytes. Con este formato se evita la realización de una decodificación de N bits durante la experimentación con las aplicaciones ad-hoc, donde N son los bits de profundidad de codificación de cada conjunto de datos de prueba.

4.2.1. Implementación del Modelo de Almacenamiento Horizontal

El modelo de almacenamiento horizontal requiere de tablas que presenten un esquema de la forma S_H=<id, ecg>. Con el fin de almacenar el identificador de cada una de las señales se utilizó un tipo de dato entero. Para el almacenamiento de la señal de ECG digital (la cual es una secuencia de números reales) se utilizó un tipo de dato arreglo que contiene números de punto flotante. El tipo de dato arreglo fue introducido en el estandar SQL:1999 [40]. Cada uno de los valores presentes dentro de una señal fue almacenado dentro de una posición del arreglo, con lo que no sólo se respetan los valores de la señal, sino también el orden en el que se encuentran. De esta manera, el esquema para cada una de las tablas en las cuales fueron almacenadas las señales de ECG digitales se estableció de la forma $S_H =< sid\,int, s\,float[] >$.

Para cada una de las cuatro bases de datos seleccionadas se creó una tabla distinta de acuerdo al esquema anterior, en las cuales fueron almacenadas sus respectivas señales de ECG digitales. Para la base de datos AAMI se creó una tabla nombrada *h_aami* en la cual se almacenaron 10 registros (10 señales con 1 derivación cada una). Para la base de datos QT se creó una tabla llamada *h_qt* en la que se almacenaron 210 registros (105 señales con 2 derivaciones cada una). Para la base de datos MIT-BIH se creó la tabla *h_mit* en la que se almacenaron 96 registros (48 señales con 2 derivaciones cada una). Finalmente, para la base de datos European se creó la tabla *h_edb* en la cual se almacenaron 180 registros (90 señales con 2 derivaciones cada una).

4.2.2. Implementación del Modelo de Almacenamiento Vertical

El modelo de almacenamiento vertical requiere que las tablas creadas presenten un esquema de la forma S_V=<id, indice, valor>. Siguiendo dicho esquema, el identificador de cada una de las señales de ECG digitales fue almacenado utilizando un tipo de dato entero. De igual manera, el índice que representa la posición de cada valor dentro de la señal de ECG digital se almacenó como un tipo de dato entero. Finalmente, cada uno de los valores dentro de las señales de ECG digitales fue almacenado como un tipo de dato numérico de punto flotante. De esta manera, el esquema para cada una de las tablas que almacenan las señales de ECG digitales en el modelo de almacenamiento vertical presenta la forma $S_V =< sid\,int, indice\,int, valor\,float >$.

De manera similar al modelo de almacenamiento horizontal, se creó una tabla para cada una de las cuatro bases de datos seleccionadas de acuerdo al esquema anterior. La utilización del modelo de almacenamiento vertical implica el manejo de un número de registros mucho mayor que los utilizados en el modelo de almacenamiento horizontal. Esto se debe a que es necesario almacenar $n \times f \times t \times d$ registros para cada una de las bases de datos, donde n es el número de señales de ECG digitales en la base de datos, f es la frecuencia de muestreo utilizada, t es la longitud en segundos de las señales de ECG digitales y d es el número de derivaciones presentes en cada señal de ECG digital.

Para la base de datos AAMI se creó una tabla denominada *v_aami* en la cual se almacenaron $10 \times 720 \times 90 \times 1$ registros aproximadamente. Para la base de datos QT se creó una tabla llamada *v_qt* en la que se almacenaron $105 \times 250 \times 900 \times 2$ registros. Para la base de datos MIT-BIH se creó la tabla *v_mit* en la que se almacenaron $48 \times 360 \times 1800 \times 2$ registros. Finalmente, para la base de datos European se creó la tabla *v_edb* donde se almacenaron $90 \times 250 \times 7200 \times 2$ registros.

4.2.3. Selección de Patrones Para los Algoritmos de Búsqueda por Semejanza

Para implementar los algoritmos propuestos de búsqueda por semejanza fue necesario crear un conjunto de patrones de interés, los cuales son buscados dentro de cada una de las señales almacenadas anteriormente. Dado que el repositorio physionet no ofrece patrones de señales de ECG digitales como parte de sus bases de datos, nosotros generamos nuestros patrones a partir de las señales de las cuatro bases de datos seleccionadas.

En la literatura se menciona que $\frac{1}{4}$ de segundo es tiempo suficiente para observar un latido cardíaco completo [18, 55]. En base a esto podemos definir que un patrón de interés es una subsecuencia s'_{ij} obtenida de una señal de ECG digital s previamente almacenada. La subsecuencia s'_{ij} comprende los valores desde la posición i hasta la posición j, donde $j - i = (\frac{1}{4})(\frac{f}{1})$, donde f es la frecuencia de muestreo de s. De esta manera, para cada una de las tablas creadas bajo el modelo de almacenamiento horizontal se creó una tabla de patrones con esquema P_H=<pid int, q float[ ]>, donde cada patrón q representa el último $\frac{1}{4}$ de segundo correspondiente a cada señal s almacenada. Posteriormente se eliminó el último $\frac{1}{4}$ de segundo de cada una de las señales s almacenadas en los modelos de almacenamiento horizontal y vertical.

Tras la aplicación de los pasos anteriores, a partir de la tabla *h_aami* se creó la tabla de patrones *h_aami_pat*, la cual contenía 10 registros con patrones de longitud $(\frac{1}{4})(\frac{720}{1}) = 180$. A partir de la tabla *h_qt* se creó la tabla de patrones *h_qt_pat*, donde se almacenaron 210 registros con patrones de longitud $(\frac{1}{4})(\frac{250}{1}) \approx 62$. De la tabla *h_mit* se creó la tabla de patrones *h_mit_pat* donde se almacenaron 96 registros con patrones de longitud $(\frac{1}{4})(\frac{360}{1}) = 90$. Finalmente, a partir de la tabla *h_edb* se generó la tabla *h_edb_pat* donde se almacenaron 180 registros representando patrones de longitud $(\frac{1}{4})(\frac{250}{1}) \approx 62$.

En la generación de patrones para la búsqueda sobre los datos bajo el mode-

lo de almacenamiento vertical fue necesario utilizar los patrones generados a partir de las tablas bajo el modelo de almacenamiento horizontal. Esto a causa de que el SGBD presenta una limitación para la creación de UDFs que involucren el uso datos provenientes de dos o más tablas bajo el esquema de almacenamiento vertical. De esta manera, los patrones utilizados fueron aquellos almacenados bajo el esquema P_H anteriormente descrito.

4.2.4. Requerimientos Físicos de Almacenamiento

Las señales de ECG digitales que aún no han sido tratadas representan la fuente de información principal para cualquier tipo de procesamiento que pueda ser aplicado sobre ellas [18]. Durante la etapa de experimentación de este trabajo de tesis se trabajó sobre señales de ECG digitales no tratadas. En otras palabras, las señales de ECG digitales se almacenaron sobre los modelos de almacenamiento propuestos horizontal y vertical tal cual son ofrecidas por parte del repositorio *Physionet*. Para el caso de las aplicaciones ad-hoc, los datos de las señales fueron almacenados sobre un formato binario para evitar la decoficicación de bits. Para tener una idea clara sobre los requerimientos en disco que cada uno de nuestros modelos de almacenamiento, así como los archivos binarios utilizados requieren, fue necesaria la revisión de las dimensiones de las tablas y los archivos binatios utilizados una vez que los datos fueron almacenados. En la tabla 4.2 se describe el espacio en disco necesario por cada una de las bases de datos utilizadas sobre los modelos de almacenamiento propuestos y archivos binarios.

Cada una de las derivaciones presentes en los conjuntos de datos fueron tomados como una señal de ECG digital independiente y almacenadas en archivos binarios separados. El identificador de cada archivo binario fue asignado al nombre de cada archivo. Cada valor presente en una señal de ECG digital fue almacenado como un dato numérico de punto flotante que requiere 4 bytes de espacio en disco. De esta

Base de Datos	Modelo de Almacenamiento		
	Horizontal	Vertical	Binarios
AAMI	0.6	27	1.8
QT	53	1,781	160
MIT-BIH	93	2,440	260
European	417	13,082	1,260

Tabla 4.2: Requerimientos Físicos de Almacenamiento de Datos en Disco (en Megabytes).

manera, por ejemplo para la base de datos AAMI el espacio en disco requerido por los archivos binarios fue de 1.8MB. Para el almacenamiento en archivos binarios de la base de datos QT se requirieron 160MB, para la base de datos MIT-BIH 220MB y finalmente para la base de datos European 1,260MB.

Para el caso particular del modelo de almacenamiento horizontal, se utilizó una tabla con esquema $< sid\,int, s\,float[] >$ para el almacenamiento de las señales de ECG digitales. Esto es, un dato de tipo entero para el identificador de la señal y un dato de tipo arreglo numérico de punto flotante para el almacenamiento de la señal en sí. En el SGBD Postgres versión 9.3 no es posible almacenar valores muy largos en una sola columna. Para resolver esta limitación, los valores muy largos son comprimidos o separados en múltiples renglones. Esto sucede de manera transparente al usuario. La técnica utilizada para poder hacer esto se denomina TOAST (The Oversized-Attribute Storage Technique) y es aplicada de manera nativa sobre el tipo de dato arreglo cuando requieren más espacio del asignado a una página, es decir 8 kiloBytes. Esto último presenta una gran ventaja para los requerimientos de disco necesarios por el modelo de almacenamiento horizontal que se propone en este trabajo. El espacio requerido por la base de datos AAMI dentro del SGBD es de 0.6MB, esto representa 3 veces menos espacio requerido con respecto a los archivos binarios. Para la base de datos QT se presenta una reducción de espacio en disco de 3 veces con respecto a lo requerido por los archivos binarios. Para la base de datos MIT-BIH se tuvo una reducción de espacio en 2.8 veces con respecto al espacio requerido por los archivos

binarios. Finalmente para la base de datos European se tuvo una mejora similar con respecto al espacio en disco requerido por los archivos binarios en una proporción de 3 veces menos espacio.

En el modelo de almacenamiento vertical se utilizó el esquema $< sid\,int, indice\,int, valor\,float >$. Esto es, un tipo de dato entero para el identificador de la señal, un dato de tipo entero para el indice de cada valor dentro de la señal y un dato numérico de punto flotante para cada uno de los valores dentro de la señal. Bajo este esquema de almacenamiento, el número de registros para poder almacenar una sola señal de ECG digital aumenta considerablemente con respecto al único registro necesario en el modelo de almacenamiento horizontal. El número de registros aumenta en orden $1 : t \times f$, donde t es la duración en segundos de la señal de ECG digital y f es la frecuencia de muestreo. El espacio requerido por la base de datos AAMI es de 27MB, lo que representa un aumento de 45 veces con respecto al espacio utilizado en el modelo de almacenamiento horizontal. Para la base de datos QT se presenta un aumento de espacio en disco de 33.6 veces con respecto a lo requerido por el modelo de almacenamiento horizontal. En el caso de la base de datos MIT-BIH se tuvo un aumento de espacio en proporción de 26.2 veces con respecto al espacio requerido por el modelo de almacenamiento horizontal. Finalmente para la base de datos European utilizó un mayor espacio en disco con respecto al espacio en disco requerido por el modelo de almacenamiento horizontal en una proporción de 31.4 veces más.

4.3. Búsqueda de Subsecuencias con Consultas SQL

En esta sección se describen las consultas utilizadas durante la evaluación experimental del presente trabajo de tesis. Las consultas desarrolladas fueron definidas como una serie de subconsultas que conforman la funcionalidad necesaria para la aplicación de los algoritmos de búsqueda por semejanza tanto para K-NN como para ϵ-search sobre las señales de ECG digitales almacenadas bajo los modelos de almacenamiento

horizontal y almacenamiento vertical. La utilización de subconsultas requiere que los datos que se usan entre ellas deben ser almacenados ya sea de manera transitoria en disco o de manera volátil en memoria principal. Con dicho fin, se desarrollaron distintas versiones de las consultas propuestas utilizando: a) tablas transitorias, b) vistas y c) CTEs (*Common Table Expressions*) como medios auxiliares para el tratamiento de los datos intermedios de las subconsultas.

4.3.1. Alternativas de Manejo de Datos Intermedios entre Subconsultas

El manejo de datos intermedios puede ser determinante para afectar directamente los tiempos de ejecución de una consulta en particular. Es por ello, que en este trabajo se realizaron experimentos sobre distintos métodos de manejo de datos intermedios entre subconsultas. Estos métodos están disponibles en el estándar SQL:1999 y se describen a continuación.

- Tablas transitorias: Una tabla transitoria nos permite almacenar de manera temporal y procesar datos intermedios, así como aplicar sobre ella las operaciones SELECT, UPDATE, JOIN, etc. que son aplicables sobre cualquier otra tabla existente en el SGBD. Algunos SGBD actuales son capaces de eliminar una tabla transitoria cuando la sesión del usuario que la creó termina. Sin embargo, en los SGBD donde eso no sucede es necesario utilizar el comando DROP TABLE <nombre_tabla> para poder eliminar la tabla transitoria creada, de otra manera quedará guardada dentro del SGBD de forma permanente. Para poder crear una tabla transitoria que almacena el resultado de una consulta se utiliza la siguiente sintaxis.

```
SELECT <a1,a2,...aN> INTO <tabla_tmp>
FROM <tabla>...;
```

Donde *<a1,a2,...aN>* es la lista de atributos a seleccionar en la consulta, *tabla_tmp* es el nombre de la tabla transitoria donde se almacenará el resultado de la consulta, y *tabla* es el nombre de la tabla de donde se selecciona la lista de atributos.

- Vistas: En términos generales, una vista es un atajo o un acceso directo para una consulta de tipo SELECT. La definición de una vista queda almacenada dentro del SGBD, sin embargo, cuando una vista se utiliza, el resultado necesitará ser generado cada vez, incluso si es ejecutada más de una vez dentro de una misma consulta. Para poder crear una vista se utiliza la siguiente sintaxis.

```
CREATE VIEW <n_vista> AS
SELECT <a1,a2,...aN>
FROM <tabla>...;
```

Donde *n_vista* es el nombre que se le da a la vista, *<a1,a2,...aN>* es la lista de atributos a seleccionar en la consulta y *tabla* es el nombre de la tabla de donde se selecciona la lista de atributos. Para poder hacer uso de una vista se utiliza una consulta de tipo SELECT en donde se especifíca el nombre de la vista a utilizar de la siguiente manera.

```
SELECT <a1,a2...aN>
FROM <n_vista>...;
```

- CTEs: A partir del estándar SQL:1999 se utiliza el comando WITH para poder definir una o más CTEs. Las CTEs son utilizadas como consultas auxiliares durante la creación de una consulta mayor. Cada CTE puede ser considerada como una tabla transitoria que existe en memoria principal sólamente durante el tiempo de ejecución de la consulta. Si el tamaño necesario para almacenar los datos generados por una CTE es mayor a la memoria disponible entonces

se realiza el intercambio de datos entre memoria principal y disco para poder obtener el resultado final. El resultado generado por cada CTE es solamente accesible por las CTEs posteriores (o para ella misma cuando utiliza recursión), y debe existir siempre una subconsulta denominada consulta principal que haga el llamado a la última CTE definida. Para poder utilizar CTEs en una consulta se requiere de la siguiente sintaxis.

```
WITH <nombre_cte1> AS
        (SELECT <a1,a2,...aN>
        FROM <tabla1>...),
WITH <nombre_cte2> AS
        (SELECT <b1,b2,...bM>
        FROM <tabla2>...)
SELECT <a1,a2...aN,b1,b2,...bM>
        FROM <nombre_cte1>,<nombre_cte2>;
```

Donde *nombre_cte1* y *nombre_cte2* son los nombres que se les da a las CTEs, *<a1,a2,...aN>* y *<b1,b2,...bM>* son las listas de atributos a seleccionar en cada CTE y, *tabla1* y *tabla2* es el nombre de las tablas de donde se obtiene la lista de atributos de cada CTE respectivamente. Finalmente, en la subconsulta principal puede seleccionarse cualquier atributo presente en el resultado de las CTEs previos mediante la realización de producto cruz u operadores JOIN entre ellas.

Con base en lo anterior, a continuación se describen las consultas que fueron utilizadas durante la etapa de experimentación en este trabajo de tesis. En la Tabla 4.3 se observa un resumen de las consultas que se proponen. Para cada una de las consultas se describe el número de subconsultas que las componen, el número de UDFs desarrolladas y el número de funciones nativas del SGBD utilizadas. El nombre asignado a cada una de las consultas presenta el siguiente formato: $m_d_a_<v>$. En este formato m representa el modelo de almacenamiento utilizado, d representa la alternativa para el manejo de datos intermedios, a representa el algoritmo de búsqueda

Abreviación	Algoritmo	Modelo de Almacenamiento	Medio de Almacenamiento Transitorio	Subconsultas Utilizadas	# de UDF Utilizadas	# de Funciones SQL:1999
H_TMP_K_V1	K-NN	Horizontal	Tablas Transitorias	4	3	1
H_TMP_K_V2	K-NN	Horizontal	Tablas Transitorias	3	2	1
H_TMP_K_V3	K-NN	Horizontal	Tablas Transitorias	3	1	2
H_TMP_K_V4	K-NN	Horizontal	Tablas Transitorias	4	2	1
H_VIEW_K_V1	K-NN	Horizontal	Vistas	4	3	1
H_VIEW_K_V2	K-NN	Horizontal	Vistas	3	2	1
H_VIEW_K_V3	K-NN	Horizontal	Vistas	3	1	2
H_VIEW_K_V4	K-NN	Horizontal	Vistas	4	2	1
H_WITH_K_V1	K-NN	Horizontal	CTE	4	3	1
H_WITH_K_V2	K-NN	Horizontal	CTE	3	2	1
H_WITH_K_V3	K-NN	Horizontal	CTE	3	1	2
H_WITH_K_V4	K-NN	Horizontal	CTE	4	2	1
V_TMP_K	K-NN	Vertical	Tablas Transitorias	1	1	0
V_VIEW_K	K-NN	Vertical	Vistas	1	1	0
V_WITH_K	K-NN	Vertical	CTE	1	1	0
H_TMP_E_V1	ϵ-search	Horizontal	Tablas Transitorias	4	3	1
H_TMP_E_V2	ϵ-search	Horizontal	Tablas Transitorias	3	2	1
H_TMP_E_V3	ϵ-search	Horizontal	Tablas Transitorias	3	1	2
H_TMP_E_V4	ϵ-search	Horizontal	Tablas Transitorias	4	2	1
H_VIEW_E_V1	ϵ-search	Horizontal	Vistas	4	3	1
H_VIEW_E_V2	ϵ-search	Horizontal	Vistas	3	2	1
H_VIEW_E_V3	ϵ-search	Horizontal	Vistas	3	1	2
H_VIEW_E_V4	ϵ-search	Horizontal	Vistas	4	2	1
H_WITH_E_V1	ϵ-search	Horizontal	CTE	4	3	1
H_WITH_E_V2	ϵ-search	Horizontal	CTE	3	2	1
H_WITH_E_V3	ϵ-search	Horizontal	CTE	3	1	2
H_WITH_E_V4	ϵ-search	Horizontal	CTE	4	2	1
V_TMP_E	ϵ-search	Vertical	Tablas Transitorias	1	1	0
V_VIEW_E	ϵ-search	Vertical	Vistas	1	1	0
V_WITH_E	ϵ-search	Vertical	CTE	1	1	0

Tabla 4.3: Resumen General de las Consultas Propuestas.

utilizado y v representa la versión de una misma consulta.

4.3.2. Consultas Sobre el Modelo de Almacenamiento Horizontal Para el Algoritmo K-NN

De acuerdo a la estructura general de consultas descritas en la Sección 3.4.1 hemos propuesto un total de 12 cosultas para la implementación del algoritmo K-NN sobre el modelo de almacenamiento horizontal. La composición de estas 12 consultas se basa en cuatro consultas principales que fueron implementadas con el uso de tres distintos medios auxiliares para el almacenamiento transitorio de datos. Cuatro de las consultas hacen uso de tablas transitorias, otras cuatro consultas hacen uso de

vistas y las últimas cuatro consultas hacen uso de CTEs. Cada una de las consultas requiere la definición de cuatro variables: S_H, P_H, X y K, donde S_H es el nombre de la tabla donde están almacenadas las señales de ECG bajo el modelo de almacenamiento horizontal, P_H es el nombre de la tabla que contiene los patrones, X es el identificador del patrón q que se busca dentro de S_H, y K es el valor de k para el algoritmo K-NN. El resultado obtenido de cada una de las consultas es una tabla con esquema $< id\,int, pos\,int, sem\,float >$, donde id es el identificador de la señal de ECG digital almacenada s_i en la cual se encontró el patrón q, pos representa la posición donde comienza s'_i dentro de s_i y sem es el valor de semejanza de la subsecuencia s'_i con respecto al patrón q. De igual manera se crearon cuatro distintas UDFs identificadas la Sección 3.4.1. Esto con el fin de lograr la integración del algoritmo K-NN sobre el modelo de almacenamiento horizontal. Estas cuatro UDFs son descritas a continuación.

- **Dist()**
  - Descripción: Función utilizada para el cálculo de semejanza basada en la distancia Euclideana entre dos arreglos numéricos de punto flotante. Ambos arreglos requieren ser de la misma longitud.
  - Parámetros de entrada: Arreglo s y arreglo q.
  - Salida: Arreglo numérico de punto flotante que contiene todos los valores de semejanza entre q y todas las posibles subsecuencias de s.

- **KminValues()**
  - Descripción: Función utilizada obtener los k valores considerados como mejores semejanzas dentro de un arreglo numérico de punto flotante.
  - Parámetros de entrada: Arreglo de semejanzas dsi y valor de k para delimitar el número de resultados.

  - Salida: Arreglo numérico de punto flotante que contiene las k mejores semejanzas encontradas en el arreglo de entrada dsi.

- **Pos()**

  - Descripción: Función utilizada para obtener la posición de la primera coincidencia de un valor numérico de punto flotante dentro de un arreglo numérico de punto flotante.
  - Parámetros de entrada: Arreglo dsi y valor de semejanza sem_val.
  - Salida: Valor numérico de tipo entero que representa la posición donde se encontró sem_val dentro del arreglo dsi.

- **KminValuesPos()**

  - Descripción: Esta función es una variante de la función **KminValues()**. Es utilizada obtener las posiciones de los k valores considerados como mejores semejanzas dentro de un arreglo numérico de punto flotante.
  - Parámetros de entrada: Arreglo de semejanzas dsi y valor de k para delimitar el número de resultados.
  - Salida: Arreglo numérico de punto flotante que contiene las posiciones de los k valores considerados como mejores semejanzas encontrados en el arreglo dsi.

En los siguientes puntos se describe a detalle cada una de las consultas propuestas para poder obtener los resultados del algoritmo K-NN sobre el modelo de almacenamiento horizontal haciendo uso de las UDFs descritas anteriormente.

Consultas Para Resolver K-NN Utilizando Tablas Transitorias Sobre el Modelo de Almacenamiento Horizontal.

- H_TMP_K_V1: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de tres UDFs: Dist(), KminValues() y Pos(), así como una función del estándar SQL:1999: unnest().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, KminValues(dsi, K) AS k_dsi INTO T2
        FROM T1;
SELECT sid, unnest(k_dsi) AS k_global INTO T3
        FROM T2
        ORDER BY k_global ASC
        LIMIT K;
SELECT T1.sid AS id, Pos(dsi, k_global) AS pos, k_global AS sem
        FROM T1,T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC;
```

La descripción de la consulta es la siguiente. La primera subconsulta representa el operador $HSim$ y genera una tabla $T1$ con esquema $< sid\,int, dsi\,float[] >$. Dado el identificador *pid* de un patrón q, esta subconsulta se utiliza para el cálculo de semejanza entre q y todas las posibles subsecuencias encontradas en s. De esta manera, para cada arreglo s en S_H la subconsulta regresa el identificador *sid* de s y un arreglo *dsi* que contiene todos los valores de semejanza calculados entre s y q. La segunda subconsulta representa al operador $HTopSeq$ y genera una tabla $T2$ con esquema $< sid\,int, k_dsi\,float[] >$. Para cada registro en $T1$, esta subconsulta obtiene el identificador *sid* y un arreglo *k_dsi* que contiene los mejores K valores encontrados en *dsi*. La tercera subconsulta realiza lo estipulado por los operadores $HSort$ y $HTop$ y genera una tabla $T3$

con esquema $< sid\,int, k_global\,float >$. Para cada registro en $T2$ esta subconsulta obtiene el identificador *sid* y utiliza la función de arreglos unnest(), para expandir un arreglo de datos en un conjunto de registros en una columna *k_global*. Una vez expandidos, ordena los registros de manera ascendiente y limita los resultados a K valores utilizando la cláusula LIMIT. Por último, la cuarta subconsulta representa el operador $HPos$ y genera una tabla con esquema $< id\,int, pos\,int, sem\,float >$. A partir de los datos en $T1$ y $T3$ esta subconsulta obtiene el identificador *sid*, la posición donde *k_global* aparece en *dsi* y el valor de *k_global*, dando el resultado final del algoritmo K-NN.

- H_TMP_K_V2: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de tres subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y Pos() así como una función del estándar SQL:1999: unnest().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, unnest(dsi) AS k_global INTO T2
        FROM T1
        ORDER BY k_global ASC
        LIMIT K;
SELECT T1.sid AS id, Pos(dsi, k_global) AS pos, k_global AS sem
        FROM T1,T2
        WHERE T1.sid = T2.sid
        ORDER BY sem ASC;
```

La descripción de la consulta se detalla a continuación. La primera subconsulta genera la tabla $T1$ y presenta el mismo funcionamiento que la primer subconsulta de H_TMP_K_V1. La segunda subconsulta representa a los operadores $HTopSeq$, $HSort$ y $HTop$. Genera una tabla $T2$ con esquema $< sid\,int, k_global\,float >$. Esta subconsulta toma de $T1$ el identificador *sid* y expande los arreglos

dsi como una sola columna *k_global*. Una vez hecha la expansión de arreglos, ordena todos los valores de la columna *k_global* de manera ascendente y limita los registros a los primeros K una vez ordenados. Por último, la tercera subconsulta representa el operador *HTop* y tiene el mismo funcionamiento que la cuarta subconsulta de H_TMP_K_V1, generando así una tabla con esquema $< id\,int, pos\,int, sem\,float >$ y representa el resultado final del algoritmo K-NN.

- H_TMP_K_V3: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de tres subconsultas en las cuales se utiliza un total de una UDF: Dist() y dos funciones del estándar SQL:1999: unnest() y row_number().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, unnest(dsi) AS k_global INTO T2
        FROM T1;
SELECT sid AS id, row_number() over
        (PARTITION BY sid) AS pos, k_global AS sem
        FROM T2
        ORDER BY sem ASC
        LIMIT K;
```

La primera subconsulta representa el operador $HSim$ y tiene el mismo funcionamiento que la primer subconsulta de H_TMP_K_V1, generando una tabla $T1$ con esquema $< sid\,int, dsi\,float[] >$. La segunda subconsulta representa parte del operador $HSort$ y genera una tabla $T2$ con esquema $< sid\,int, k_global\,float >$. Esta subconsulta selecciona de $T1$ el identificador *sid* y expande los arreglos *dsi* como una sola columna *k_global*. Finalmente, una tercera subconsulta representa el funcionamiento de los operadores $HTopSeq$, $HSort$, $HTop$ y $HPos$, generando una tabla con esquema $< id\,int, pos\,int, sem\,float >$. Esta subconsulta obtiene de $T2$ el identificador *sid*. Para obtener el valor de *pos* es necesario

ordenar el total de valores *k_global* en orden ascentende y posteriormente se utiliza una función de ventana llamada row_number(). Esta función de ventana asigna un identificador único a cada elemento en un conjunto de registros. Como parámetro para la partición de registros se utiliza el identificador *sid*, mientras que para definir el tamaño de la ventana se toma como valor defaultt la longitud de s. El valor de *sem* se toma también de $T2$, para finalmente delimitar a K el número de registros para generar la tabla final, que representa el resultado del algoritmo K-NN.

- H_TMP_K_V4: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y KminValuesPos() así como una función del estándar SQL:1999: unnest().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, KminValuesPos(dsi, K) AS k_dsi INTO T2
        FROM T1;
SELECT sid, unnest(k_dsi) AS k_global INTO T3
        FROM T2;
SELECT T1.sid AS id, T3.k_global AS pos, T1.dsi[T3.k_global] AS sem,
        FROM T1, T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC
        LIMIT K;
```

La primera subconsulta, al igual que la primera subconsulta de H_TMP_K_V1 corresponde el operador $HSim$ y presenta el mismo comportamiento. La segunda subconsulta representa una versión alternativa del operador $HTopSeq$ y genera una tabla $T2$ con esquema $< sid\,int, k_dsi\,float[] >$. Para cada registro en $T1$ obtiene el identificador *sid* y el arreglo *k_dsi* compuesto por las posiciones de los K mejores valores encontrados en *dsi*. La tercera subconsulta

corresponde en parte al operador $HSort$ y se encarga de generar una tabla $T3$ con esquema $< sid\,int, k_global\,float >$. A partir de $T2$ obtiene el identificador sid y expande los arreglos k_dsi dentro de una columna k_global. Finalmente, una cuarta subconsulta representa a los operadores $HTopSeq$, $HSort$, $HTop$ y $HPos$, generando una tabla con esquema $< id\,int, pos\,int, sem\,float >$. A partir de los resultados de $T1$ y $T3$ esta subconsulta obtiene el identificador id, la posición pos y gracias a la facilidad ofrecida por Postgres podemos obtener el valor de un arreglo de datos dado un índice. Finalmente los valores son ordenados de forma ascendente respecto al valor de sem y se limita el número de resultados a K utilizando la cláusula LIMIT. Con esto se genera el resultado que representa la salida del algoritmo K-NN.

Consultas Para Resolver K-NN Utilizando Vistas Sobre el Modelo de Almacenamiento Horizontal.

- H_VIEW_K_V1: Esta consulta utiliza vistas para el tratamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de tres UDFs: Dist(), KminValues() y Pos(), así como una función del estándar SQL:1999: unnest(). En términos generales, la descripción de esta consulta es idéntica a la descripción de la consulta H_TMP_K_V1. De igual manera, presenta el mismo el número de subconsultas así como los resultados que cada una de ellas presentan. La diferencia radica en el medio para el tratamiento de datos intermedios.

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, KminValues(dsi, K) AS k_dsi
        FROM T1;
CREATE VIEW T3 AS SELECT sid, unnest(k_dsi) AS k_global
        FROM T2
        ORDER BY k_global ASC
        LIMIT K;
SELECT T1.sid AS id, Pos(dsi, k_global) AS pos, k_global AS sem
        FROM T1,T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC;
```

- H_VIEW_K_V2: Esta consulta es homóloga a la consulta H_TMP_K_V2, salvo que utiliza vistas para el tratamiento de datos intermedios en lugar de tablas transitorias. Está conformada a su vez por una serie de tres subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y Pos() así como una función del estándar SQL:1999: unnest().

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, unnest(dsi) AS k_global
        FROM T1
        ORDER BY k_global ASC
        LIMIT K;
SELECT T1.sid AS id, Pos(dsi, k_global) AS pos, k_global AS sem
        FROM T1,T2
        WHERE T1.sid = T2.sid
        ORDER BY sem ASC;
```

- H_VIEW_K_V3: Esta consulta es una versión homóloga a la consulta H_TMP_K_V3. La diferencia está en que utiliza vistas para el tratamiento de datos intermedios. La consulta Está conformada a su vez por una serie de tres subconsultas en las cuales se utiliza un total de una UDF: Dist() y dos funciones del estándar

SQL:1999: unnest() y row_number().

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, unnest(dsi) AS k_global
        FROM T1;
SELECT sid AS id, row_number() over
        (PARTITION BY sid) AS pos, k_global AS sem
        FROM T2
        ORDER BY sem ASC
        LIMIT K;
```

- H_VIEW_K_V4: Esta consulta utiliza vistas para el tratamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y KminValuesPos() así como una función del estándar SQL:1999: unnest(). Presenta el mismo número de subconsultas así como la lógica de la consulta H_TMP_K_V4.

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, KminValuesPos(dsi, K) AS k_dsi
        FROM T1;
CREATE VIEW T3 AS SELECT sid, unnest(k_dsi) AS k_global
        FROM T2;
SELECT T1.sid AS id, T3.k_global AS pos, T1.dsi[T3.k_global] AS sem,
        FROM T1, T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC
        LIMIT K;
```

Consultas Para Resolver K-NN Utilizando CTEs Sobre el Modelo de Almacenamiento Horizontal.

- H_WITH_K_V1: Esta consulta utiliza CTEs para el tratamiento de datos intermedios y es homóloga a las consultas H_TMP_K_V1 y H_VIEW_K_V1. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de tres UDFs: Dist(), KminValues() y Pos() así como la función del estándar SQL:1999: unnest().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS (SELECT sid, KminValues(dsi, K) AS k_dsi
        FROM T1),
T3 AS (SELECT sid, unnest(k_dsi) AS k_global
        FROM T2
        ORDER BY k_global ASC
        LIMIT K)
SELECT T1.sid AS id, Pos(dsi, k_global) AS pos, k_global AS sem
        FROM T1,T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC;
```

- H_WITH_K_V2: Esta consulta es homóloga a las consultas H_TMP_K_V2 y H_VIEW_K_V2. A diferencia de estas últimas, H_WITH_K_V2 utiliza CTEs para el tratamiento de datos intermedios. Está conformada a su vez por una serie de tres subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y Pos() así como una función del estándar SQL:1999: unnest().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS (SELECT sid, unnest(dsi) AS k_global
        FROM T1
        ORDER BY k_global ASC
        LIMIT K)
SELECT T1.sid AS id, Pos(dsi, k_global) AS pos, k_global AS sem
        FROM T1,T2
        WHERE T1.sid = T2.sid
        ORDER BY sem ASC;
```

- H_WITH_K_V3: Esta consulta utiliza CTEs para el tratamiento de datos intermedios y es homóloga a las consultas H_TMP_K_V3 y H_VIEW_K_V3. Está conformada a su vez por una serie de tres subconsultas en las cuales se utiliza un total de una UDF: Dist() y dos funciones del estándar SQL:1999: unnest() y row_number().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS (SELECT sid, unnest(dsi) AS k_global
        FROM T1)
SELECT sid AS id, row_number() over
        (PARTITION BY sid) AS pos, k_global AS sem
        FROM T2
        ORDER BY sem ASC
        LIMIT K;
```

- H_WITH_K_V4: Esta consulta es homóloga a las consultas H_TMP_K_V4 y H_VIEW_K_V4 pero utiliza CTEs para el tratamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y KminValuesPos() así como una función del estándar SQL:1999: unnest().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS (SELECT sid, KminValuesPos(dsi, K) AS k_dsi
        FROM T1),
T3 AS (SELECT sid, unnest(k_dsi) AS k_global
        FROM T2)
SELECT T1.sid AS id, T3.k_global AS indice, T1.dsi[T3.k_global] AS valor,
        FROM T1, T3
        WHERE T1.sid = T3.sid
        ORDER BY valor
        LIMIT K;
```

4.3.3. Consultas Sobre el Modelo de Almacenamiento Vertical Para el Algoritmo K-NN

Con base a la metodología descrita en la Sección 3.4.2 hemos propuesto un total de tres cosultas para la implementación del algoritmo K-NN sobre el modelo de almacenamiento vertical. La composición de estas tres consultas se basa en una consulta principal la cual fue implementada con el uso de tres distintos medios auxiliares para el almacenamiento transitorio de datos. La primera consulta hace uso de tablas transitorias, la segunda consulta hace uso de vistas y tercer consulta hace uso de CTEs. Cada una de estas consultas requiere la definición de cuatro variables: S_V, P_H, X y K, donde S_V es el nombre de la tabla donde están almacenadas las señales de ECG bajo el modelo de almacenamiento vertical, P_H es el nombre de la tabla que contiene los patrones, X es el identificador del patrón q que se busca dentro de S_V, y K es el valor de k para el algoritmo K-NN. El resultado obtenido de cada una de las consultas debe ser el mismo que el que obtenemos sobre el modelo de almacenamiento horizontal, es decir, una tabla con esquema $< id\,int, pos\,int, sem\,float >$, donde id es el identificador de la señal de ECG digital almacenada s_i donde se encontró el patrón q, pos representa la posición donde comienza s'_i dentro de s_i y sem corresponde al valor de semejanza de la subsecuencia s'_i con respecto al patrón q. Para implementar

el algoritmo K-NN sobre el modelo de almacenamiento vertical se desarrolló una UDF identificada en la Sección 3.4.2. Esta UDF es descrita a continuación.

- **v_Dist()**
  - Descripción: Esta es una función de ventana que representa el equivalente a la función $Dist()$ sobre el modelo de almacenamiento horizontal. La función $v_Dist()$ es utilizada para el cálculo de semejanza basada en la distancia Euclideana entre un conjunto de valores numéricos de punto flotante pertenecientes a una misma columna y un arreglo numérico de punto flotante el cual representa el patrón a buscar. La cardinalidad del conjunto y la longitud del arreglo deben ser iguales.
  - Parámetros de entrada: Conjunto de valores s, columna para el particionamiento de los datos *sid*, el tamaño de la ventana (longitud de la partición por default), columna para el ordenamiento de datos dentro de una partición *indice* y un arreglo q.
  - Salida: Valor escalar numérico de punto flotante que indica el valor de semejanza basado en distancia euclideana entre el patrón q y un conjunto de valores numéricos de punto flotante pertenecientes a una misma columna.

A continuación se describen a detalle cada una de las consultas propuestas para poder obtener los resultados del algoritmo K-NN sobre el modelo de almacenamiento vertical haciendo uso también de la UDF desarrollada y mencionada anteriormente.

- V_TMP_K: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada por una serie de dos subconsultas en las cuales se utiliza un total de una UDF: $v_Dist()$ y no fue necesaria alguna función del estándar SQL:1999.

```
SELECT sid, indice,v_Dist(valor, q) OVER
    (PARTITION BY sid ORDER BY indice) AS sem INTO T1
    FROM S_V, P_H
    WHERE pid = X
    ORDER BY sem ASC;
SELECT sid AS id, indice AS pos, sem
    FROM T1
    LIMIT K;
```

La descripción de la consulta es la siguiente. La primera subconsulta representa los operadores $VSim$ y $VSort$. Genera una tabla $T1$ con esquema $< sid\,int, indice\,int, sem\,float >$. Esta subconsulta utiliza la función de ventana $v_Dist()$ para el cálculo entre un conjunto de valores s pertenecientes a la columna *valor* y un arreglo numérico q. Los parámetros requeridos por una UDF de ventana fueron descritos en la Sección 3.3. La función $v_Dist()$ necesita como parámetro el valor para particionar los datos. Para este caso en particular los datos se particionan de acuerdo al identificador *sid* asociado a cada señal de ECG digital s almacenada. Dentro de la partición, los datos se ordenan de acuerdo al valor de *indice*, el cuál a su vez representa la posición asignada a *valor* dentro de s. Finalmente la segunda subconsulta representa el operador $VTop$ y genera una tabla con esquema $< id\,int, pos\,int, sem\,float >$. Esta subconsulta selecciona de $T1$ el identificador *id*, la posición donde comienza la subsecuencia s' y el valor de semejanza calculado, limitando el número de resultados a K registros. Con esto se obtiene el resultado esperado del algoritmo K-NN.

- V_VIEW_K: Esta consulta es homóloga a la consulta V_TMP_K. A diferencia de esta última, V_VIEW_K utiliza vistas para el almacenamiento de datos intermedios. Está conformada por una serie de dos subconsultas en las cuales se utiliza un total de una UDF: v_Dist() y no fue necesaria alguna función del estándar SQL:1999.

```
CREATE VIEW T1 AS SELECT sid, indice ,v_Dist(valor, q) OVER
    (PARTITION BY sid ORDER BY indice) AS sem
    FROM S_V, P_H
    WHERE pid = X
    ORDER BY sem ASC;
SELECT sid AS id, indice AS pos, sem
    FROM T1
    LIMIT K;
```

- V_WITH_K: Esta consulta es homóloga a las consultas V_TMP_K y V_VIEW_K. A diferencia de estas últimas utiliza CTEs para el almacenamiento de datos intermedios. Está conformada por una serie de dos subconsultas en las cuales se utiliza un total de una UDF v_Dist() y no fue necesaria alguna función del estándar SQL:1999.

```
WITH T1 AS (SELECT sid, indice ,v_Dist(valor, q) OVER
    (PARTITION BY sid ORDER BY indice) AS sem
    FROM S_V, P_H
    WHERE pid = X
    ORDER BY sem ASC)
SELECT sid AS id, indice AS pos, sem
    FROM T1
    LIMIT K;
```

4.3.4. Consultas Sobre el Modelo de Almacenamiento Horizontal Para el Algoritmo ϵ-Search

Siguiendo lo descrito en la Sección 3.4.3 proponemos un total de 12 consultas para la implementación del algoritmo de búsqueda ϵ-search sobre el modelo de almacenamiento horizontal. Estas consultas fueron desarrolladas sobre cuatro consultas principales implementadas con el uso de tres distintos medios auxiliares para el almacenamiento transitorio de datos. De esta manera cuatro consultas hacen uso de tablas transitorias, cuatro consultas hacen uso de vistas y cuatro consultas hacen

uso de CTEs. Cada una de las consultas requiere la definición de cuatro variables: S_H, P_H, X y ϵ, donde S_H es el nombre de la tabla donde están almacenadas las señales de ECG bajo el modelo de almacenamiento horizontal, P_H es el nombre de la tabla que contiene los patrones, X es el identificador del patrón q que se busca dentro de S_H y ϵ es el valor de umbral requerido por el algoritmo de búsqueda ϵ-search. El resultado obtenido de cada una de las consultas es una tabla con esquema $< id\,int, pos\,int, sem\,float >$, donde id representa el identificador de la señal de ECG digital almacenada s_i donde se encontró el patrón q, pos la posición donde comienza s'_i dentro de s_i y sem es el valor de semejanza de la subsecuencia s'_i con respecto al patrón q.

Para lograr la implementación del algoritmo de búsqueda ϵ-search sobre el modelo de almacenamiento vertical se reutilizaron las funciones $Dist()$ y $Pos()$ usadas en el algoritmo de búsqueda K-NN sobre el modelo de almacenamiento horizontal. De igual manera se requirió la implementación de dos nuevas UDFs para el algoritmo de búsqueda ϵ-search en específico sobre el modelo de almacenamiento horizontal. Estas dos UDFs se describen a continuación.

- **EminValues()**
  - Descripción: Esta función es utilizada para obtener todos aquellos valores de semejanza menores a un umbral ϵ encontrados dentro de un arreglo numérico de punto flotante.
  - Parámetros de entrada: Arreglo de semejanzas dsi y valor de umbral ϵ.
  - Salida: Arreglo numérico de punto flotante que contiene todos aquellos valores en dsi menores al umbral ϵ.
- **EminValuesPos()**
  - Descripción: Esta función es una variante de la función **EminValues()**. Es utilizada obtener las posiciones de los valores de semejanza menores a

un umbral ϵ encontrados dentro de un arreglo numérico de punto flotante.

- Parámetros de entrada: Arreglo de semejanzas *dsi* y valor de umbral ϵ para delimitar el número de resultados.
- Salida: Arreglo numérico de punto flotante que contiene las posiciones de los valores menores al umbral ϵ encontrados en *dsi*.

A continuación se describen a detalle las consultas propuestas para poder obtener los resultados del algoritmo de búsqueda ϵ-search sobre el modelo de almacenamiento horizontal utilizando las UDFs descritas anteriormente.

Consultas Para Resolver ϵ-Search Utilizando Tablas Transitorias Sobre el Modelo de Almacenamiento Horizontal.

- H_TMP_E_V1: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utiliza un total de tres UDFs: Dist(), EminValues() y Pos(), así como una función del estándar SQL:1999: unnest().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, EminValues(dsi, ε) AS e_dsi INTO T2
        FROM T1;
SELECT sid, unnest(e_dsi) AS e_global INTO T3
        FROM T2;
SELECT T1.sid AS id, Pos(dsi, e_global) AS pos, e_global AS sem
        FROM T1,T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC;
```

La descripción de la consulta es la siguiente. La primera subconsulta representa el operador $HSim$ y genera una tabla $T1$ con esquema $< sid\,int, dsi\,float[] >$. Dado el identificador *pid* de un patrón q, esta subconsulta se utiliza para el

cálculo de semejanza entre q y todas las posibles subsecuencias encontradas en s. De esta manera, para cada arreglo s en S_H la subconsulta regresa el identificador sid de s y un arreglo dsi que contiene todos los valores de semejanza calculados entre s y q. La segunda subconsulta representa al operador $HThresh$ y genera una tabla $T2$ con esquema $< sid\,int, e_dsi\,float[] >$. Para cada registro en $T1$, esta subconsulta obtiene el identificador sid y un arreglo e_dsi que contiene todos aquellos valores menores al umbral ϵ encontrados en dsi. La tercera subconsulta genera una tabla $T3$ con esquema $< sid\,int, e_global\,float >$. Para cada registro en $T2$ esta subconsulta obtiene el identificador sid y utiliza la función de arreglos unnest(), para expandir un arreglo de datos en un conjunto de registros en una columna e_global. Por último, la cuarta subconsulta representa los operadores $HSort$ y $HPos$, generando una tabla con esquema $< id\,int, pos\,int, sem\,float >$. A partir de los datos en $T1$ y $T3$ esta subconsulta obtiene el identificador sid, la posición donde e_global aparece en dsi y el valor de e_global, dando el resultado final del algoritmo ϵ-search.

- H_TMP_E_V2: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de tres subconsultas en las cuales se utiliza un total de dos UDFs: Dist() y Pos() así como una función del estándar SQL:1999: unnest().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, unnest(dsi) AS e_global INTO T2
        FROM T1
        ORDER BY e_global ASC;
SELECT T1.sid AS id, Pos(dsi, e_global) AS pos, e_global AS sem
        FROM T1,T2
        WHERE T1.sid = T2.sid AND e_global <= ε;
```

La descripción de la consulta se detalla a continuación. La primera subconsulta

genera la tabla $T1$ y presenta el mismo funcionamiento que la primer subconsulta de H_TMP_E_V1. La segunda subconsulta representa el operador $HSort$ y genera una tabla $T2$ con esquema $< sid\,int, e_global\,float >$. Esta subconsulta toma de $T1$ el identificador sid y expande los arreglos dsi como una columna e_global. Una vez hecha la expansión de arreglos, ordena todos los valores de la columna e_global de manera ascendente. Por último, la tercera subconsulta representa los operadores $HPos$ y $HThresh$. Esta subconsulta genera una tabla con esquema $< id\,int, pos\,int, sem\,float >$. A partir de los datos en $T1$ y $T2$ obtiene el identificador sid, la posición donde e_global se encuentra en dsi y el valor de e_global. De los registros resultantes realiza un filtro preservando sólo aquellos valores menores al umbral establecido ϵ. La tabla $T3$ representa el resultado final del algoritmo ϵ-search.

- H_TMP_E_V3: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de tres subconsultas en las cuales se utiliza un total de una UDF: Dist() y dos funciones del estándar SQL:1999: unnest() y row_number().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, unnest(dsi) AS e_global INTO T2
        FROM T1
        ORDER BY e_global ASC;
SELECT sid AS id, row_number() over
        (PARTITION BY sid) AS pos, e_global AS sem
        FROM T2
        WHERE e_global <= ε;
```

La descripción de la consulta es la siguiente. La primera subconsulta representa el operador $HSim$ y tiene el mismo funcionamiento que la primer subconsulta en H_TMP_E_V1, generando una tabla $T1$ con esquema $< sid\,int, dsi\,float[] >$.

La segunda subconsulta presenta el mismo funcionamiento que la segunda subconsulta de H_TMP_E_V2 y genera una tabla $T2$ con esquema $< sid\,int, e_global\ float >$. Finalmente, la tercera subconsulta representa el funcionamiento de los operadores $HPos$ y $HThresh$, generando una tabla con esquema $< id\,int, pos\ int, sem\ float >$. Esta subconsulta obtiene de $T2$ el identificador *sid*. Para obtener el valor de *pos* se utiliza la función de ventana row_number(). Esta función de ventana asigna un identificador único a cada elemento en un conjunto de registros. Como parámetro para la partición de registros se utiliza el identificador *sid*, mientras que para definir el tamaño de la ventana se toma como valor defaultt la longitud de s. El valor de *sem* se toma también de $T2$, para finalmente delimitar el número de resultados utilizando un filtro sobre los registros manteniendo solamente aquellos cuyo valor de *e_global* sea menor al umbral establecido ϵ. Esta salida representa el resultado final del algoritmo ϵ-search.

- H_TMP_E_V4: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utiliza un total de dos UDFs: Dist() y EminValuesPos() así como una función del estándar SQL:1999: unnest().

```
SELECT sid, Dist(s, q) AS dsi INTO T1
        FROM S_H, P_H
        WHERE pid = X;
SELECT sid, EminValuesPos(dsi, ε) AS e_dsi INTO T2
        FROM T1;
SELECT sid, unnest(e_dsi) AS e_global INTO T3
        FROM T2
        ORDER BY e_global ASC;
SELECT T1.sid AS id, T3.e_global AS pos, T1.dsi[T3.e_global] AS sem,
        FROM T1, T3
        WHERE T1.sid = T3.sid;
```

La descripción de la consulta es la siguiente. La primera subconsulta, al igual

que la primera subconsulta de H_TMP_E_V1 corresponde el operador $HSim$ y presenta el mismo comportamiento. La segunda subconsulta representa una versión alternativa del operador $HThresh$ y genera una tabla $T2$ con esquema $< sid\,int, e_dsi\,float[] >$. Para cada registro en $T1$ obtiene el identificador sid y el arreglo e_dsi compuesto por las posiciones de los valores encontrados en dsi menores al umbral establecido ϵ. La tercera subconsulta corresponde al operador $HSort$ y se encarga de generar una tabla $T3$ con esquema $< sid\,int, e_global\,float >$. A partir de $T2$ obtiene el identificador sid y expande los arreglos e_dsi dentro de una misma columna e_global. Finalmente, una cuarta subconsulta representa al operador $HPos$, generando una tabla con esquema $< id\,int, pos\,int, sem\,float >$. A partir de los resultados de $T1$ y $T3$ esta subconsulta obtiene el identificador id, la posición pos y gracias a la facilidad ofrecida por el SGBD Postgres podemos obtener el valor de un arreglo de datos dado un índice. Con esto se genera la salida que representa el resultado final del algoritmo ϵ-search.

Consultas Para Resolver ϵ-Search Utilizando Vistas Sobre el Modelo de Almacenamiento Horizontal.

- H_VIEW_E_V1: Esta consulta utiliza vistas para el tratamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de tres UDFs: Dist(), EminValues() y Pos(), así como una función del estándar SQL:1999: unnest(). En términos generales el número de subconsultas así como los resultados de cada una de ellas presentan el mismo funcionamiento que las encontradas en la consulta H_TMP_E_V1. La diferencia radica en el medio para el tratamiento de datos intermedios.

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, EminValues(dsi, ε) AS e_dsi
        FROM T1;
CREATE VIEW T3 AS SELECT sid, unnest(e_dsi) AS e_global
        FROM T2;
SELECT T1.sid AS id, Pos(dsi, e_global) AS pos, e_global AS sem
        FROM T1,T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC;
```

- H_VIEW_E_V2: Esta consulta es homóloga a la consulta H_TMP_E_V2, salvo que utiliza vistas para el tratamiento de datos intermedios en lugar de tablas transitorias. Está conformada a su vez por una serie de tres subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y Pos() así como una función del estándar SQL:1999: unnest().

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, unnest(dsi) AS e_global
        FROM T1
        ORDER BY e_global ASC;
SELECT T1.sid AS id, Pos(dsi, e_global) AS pos, e_global AS sem
        FROM T1,T2
        WHERE T1.sid = T2.sid AND e_global <= ε;
```

- H_VIEW_E_V3: Esta consulta es una versión homóloga a la consulta H_TMP_E_V3. La diferencia está en que utiliza vistas para el tratamiento de datos intermedios. La consulta Está conformada a su vez por una serie de tres subconsultas en las cuales se utiliza un total de una UDF: Dist() y dos funciones del estándar SQL:1999: unnest() y row_number().

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, unnest(dsi) AS e_global
        FROM T1
        ORDER BY e_global ASC;
SELECT sid AS id, row_number() over
        (PARTITION BY sid) AS pos, e_global AS sem
        FROM T2
        WHERE e_global <= ε;
```

- H_VIEW_E_V4: Esta consulta utiliza vistas para el tratamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y EminValuesPos() así como una función del estándar SQL:1999: unnest(). Presenta el mismo número de subconsultas así como la lógica de la consulta H_TMP_E_V4.

```
CREATE VIEW T1 AS SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X;
CREATE VIEW T2 AS SELECT sid, EminValuesPos(dsi, ε) AS e_dsi
        FROM T1;
CREATE VIEW T3 AS SELECT sid, unnest(e_dsi) AS e_global
        FROM T2
        ORDER BY e_global ASC;
SELECT T1.sid AS id, T3.e_global AS pos, T1.dsi[T3.e_global] AS sem,
        FROM T1, T3
        WHERE T1.sid = T3.sid;
```

Consultas Para Resolver ϵ-Search Utilizando CTEs Sobre el Modelo de Almacenamiento Horizontal.

- H_WITH_E_V1: Esta consulta utiliza CTEs para el tratamiento de datos intermedios y es homóloga de las consultas H_TMP_E_V1 y H_VIEW_E_V1. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan

un total de tres UDFs: Dist(), EminValues() y Pos() así como la función del estándar SQL:1999: unnest().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS (SELECT sid, EminValues(dsi, ε) AS e_dsi
        FROM T1),
T3 AS (SELECT sid, unnest(e_dsi) AS e_global
        FROM T2)
SELECT T1.sid AS id, Pos(dsi, e_global) AS pos, e_global AS sem
        FROM T1,T3
        WHERE T1.sid = T3.sid
        ORDER BY sem ASC;
```

- H_WITH_E_V2: Esta consulta es homóloga a las consultas H_TMP_E_V2 y H_VIEW_E_V2. A diferencia de estas últimas, H_WITH_E_V2 utiliza CTEs para el tratamiento de datos intermedios. Está conformada a su vez por una serie de tres subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y Pos() así como una función del estándar SQL:1999: unnest().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS SELECT sid, unnest(dsi) AS e_global
        FROM T1
        ORDER BY e_global ASC)
SELECT T1.sid AS id, Pos(dsi, e_global) AS pos, e_global AS sem
        FROM T1,T2
        WHERE T1.sid = T2.sid AND e_global <= ε;
```

- H_WITH_E_V3: Esta consulta utiliza CTEs para el tratamiento de datos intermedios y es homóloga a las consultas H_TMP_E_V3 y H_VIEW_E_V3. Está conformada a su vez por una serie de tres subconsultas en las cuales se utiliza un total de una UDF: Dist() y dos funciones del estándar SQL:1999: unnest() y

row_number().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS (SELECT sid, unnest(dsi) AS e_global
        FROM T1
        ORDER BY e_global ASC)
SELECT sid AS id, row_number() over
        (PARTITION BY sid) AS pos, e_global AS sem
        FROM T2
        WHERE e_global <= ϵ;
```

- H_WITH_E_V4: Esta consulta es homóloga a las consultas H_TMP_K_V4 y H_VIEW_K_V4 pero utiliza CTEs para el tratamiento de datos intermedios. Está conformada a su vez por una serie de cuatro subconsultas en las cuales se utilizan un total de dos UDFs: Dist() y KminValuesPos() así como una función del estándar SQL:1999: unnest().

```
WITH T1 AS (SELECT sid, Dist(s, q) AS dsi
        FROM S_H, P_H
        WHERE pid = X),
T2 AS (SELECT sid, EminValuesPos(dsi, ϵ) AS e_dsi
        FROM T1),
T3 AS SELECT sid, unnest(e_dsi) AS e_global
        FROM T2
        ORDER BY e_global ASC)
SELECT T1.sid AS id, T3.e_global AS pos, T1.dsi[T3.e_global] AS sem,
        FROM T1, T3
        WHERE T1.sid = T3.sid;
```

4.3.5. Consultas Sobre el Modelo de Almacenamiento Vertical Para el Algoritmo ϵ-search

De acuerdo a las bases descritas en la Sección 3.4.4 hemos propuesto un total de tres cosultas para la implementación del algoritmo ϵ-search sobre el modelo de almacenamiento vertical. La composición de estas tres consultas se basa en una consulta principal la cual fue implementada con el uso de tres distintos medios auxiliares para el almacenamiento transitorio de datos. La primera consulta hace uso de tablas transitorias, la segunda consulta hace uso de vistas y tercer consulta hace uso de CTEs. Cada una de estas consultas requiere la definición de cuatro variables: S_V, P_H, X y ϵ, donde S_H es el nombre de la tabla donde están almacenadas las señales de ECG bajo el modelo de almacenamiento vertical, P_H es el nombre de la tabla que contiene los patrones, X es el identificador del patrón q que se busca dentro de S_V y ϵ es el valor de umbral requerido por el algoritmo de búsqueda ϵ-search. El resultado obtenido de cada una de las consultas debe ser el mismo que el que obtenemos sobre el modelo de almacenamiento horizontal, es decir, una tabla con esquema $< id\,int, pos\,int, sem\,float >$, donde id es el identificador de la señal de ECG digital almacenada s_i donde se encontró el patrón q, pos representa la posición donde comienza s'_i dentro de s_i y sem corresponde al valor de semejanza de la subsecuencia s'_i con respecto al patrón q. Para implementar el algoritmo ϵ-search sobre el modelo de almacenamiento vertical se reutilizó la UDF $v_Dist()$ descrita en la Sección 4.3.3.

A continuación se describen a detalle cada una de las consultas propuestas para poder obtener los resultados del algoritmo ϵ-search sobre el modelo de almacenamiento vertical.

- V_TMP_E: Esta consulta utiliza tablas tansitorias para el almacenamiento de datos intermedios. Está conformada por una serie de dos subconsultas en las cuales se utiliza un total de una UDF: $v_Dist()$ y no fue necesaria alguna función del estándar SQL:1999.

```
SELECT sid, indice,v_Dist(valor, q) OVER
    (PARTITION BY sid ORDER BY indice) AS sem INTO T1
    FROM S_V, P_H
    WHERE pid = X
    ORDER BY sem ASC;
SELECT sid AS id, indice AS pos, sem
    FROM T1
    WHERE sem <= ε;
```

  La descripción de la consulta es la siguiente. La primera subconsulta representa los operadores $VSim$ y $VSort$. Genera una tabla $T1$ con esquema $< sid\,int, indice\,int, sem\,float >$. Esta subconsulta utiliza la función $v_Dist()$ para el cálculo entre un conjunto de valores s pertenecientes a la columna *valor* y un arreglo numérico q. La función $v_Dist()$ necesita como parámetro el valor para particionar los datos. Para este caso en particular los datos se particionan de acuerdo al identificador *sid* asociado a cada señal de ECG digital s almacenada. Dentro de la partición, los datos se ordenan de acuerdo al valor de *indice*, el cuál a su vez representa la posición asignada a *valor* dentro de s. Finalmente la segunda subconsulta representa el operador $VThresh$ y genera una tabla con esquema $< id\,int, pos\,int, sem\,float >$. Esta subconsulta selecciona de $T1$ el identificador *sid*, la posición donde comienza la subsecuencia s' y el valor de semejanza calculado, limitando el número de resultados a K registros. Con esto se obtiene el resultado esperado del algoritmo K-NN.

- V_VIEW_E: Esta consulta es homóloga a la consulta V_TMP_E. A diferencia de esta última, V_VIEW_K utiliza vistas para el almacenamiento de datos in-

termedios. Está conformada por una serie de dos subconsultas en las cuales se utiliza un total de una UDF: v_Dist() y no fue necesaria alguna función del estándar SQL:1999.

```
CREATE VIEW T1 AS SELECT sid, indice,v_Dist(valor, q) OVER
    (PARTITION BY sid ORDER BY indice) AS sem
    FROM S_V, P_H
    WHERE pid = X
    ORDER BY sem ASC;
SELECT sid AS id, indice AS pos, sem
    FROM T1
    WHERE sem <= ϵ;
```

- V_WITH_E: Esta consulta es homóloga a las consultas V_TMP_E y V_VIEW_E. A diferencia de estas últimas utiliza CTEs para el almacenamiento de datos intermedios. Está conformada por una serie de dos subconsultas en las cuales se utiliza un total de una UDF v_Dist() y no fue necesaria alguna función del estándar SQL:1999.

```
WITH T1 AS SELECT sid, indice,v_Dist(valor, q) OVER
    (PARTITION BY sid ORDER BY indice) AS sem INTO T1
    FROM S_V, P_H
    WHERE pid = X
    ORDER BY sem ASC)
SELECT sid AS id, indice AS pos, sem
    FROM T1
    WHERE sem <= ϵ;
```

4.3.6. Ejecución de Consultas Sobre el SGBD

Una vez definidas las consultas descritas en las secciones anteriores, el paso siguiente fue su ejecución dentro del SGBD para poder analizar y comparar la eficiencia que cada una de ellas presenta sobre su respectivo modelo de almacenamiento. Los

tiempos de ejecución obtenidos para cada una de las consultas propuestas, así como los de las aplicaciones K_ADHOC y E_ADHOC fueron seleccionados de acuerdo a la ecuación 4.1. Donde $t(S, q_j)$ es una función que determina el tiempo total en segundos que toma buscar el patrón q_j dentro de un conjunto de señales de ECG digitales S, y M es la cardinalidad del conjunto de patrones P.

$$Tiempo = \frac{\sum_{j=1}^{M} t(S, q_j)}{M} \tag{4.1}$$

Durante el cálculo de tiempos de ejecución, fue necesaria la limpieza de caché tanto para el SGBD entre consultas como para el sistema operativo durante ejecuciones de las aplicaciones ad-hoc. Para limpiar la memoria caché del SGBD y también del sistema operativo se utilizó el comando */proc/sys/vm/drop_caches* entre la ejecución de dos consultas consecutivas.

Cada una de las consultas propuestas fue ejecutada sobre los cuatro conjuntos de datos descritos en la Sección 4.1. De manera similar las aplicaciones K_ADHOC y E_ADHOC fueron ejecutadas sobre los conjuntos de datos de prueba almacenados como archivos binarios con el fin de tener un punto de comparación contra nuestra propuesta.

Las variables S_H, S_V y P_H fueron definidas de acuerdo al conjunto de datos utilizado por cada consulta para un modelo de almacenamiento en particular. Por ejemplo, para las consultas ejecutadas sobre el conjunto de datos AAMI sobre el modelo de almacenamiento horizontal se definieron las variables S_H y P_H como *h_aami* y *h_aami_pat* respectivamente. Para las consultas ejecutadas sobre el conjunto de datos QT con el modelo de almacenamiento vertical se definieron las variables S_V y P_H como *v_qt* y *h_qt* respectivamente. Por otro lado, el valor de X fue definido a partir del patrón seleccionado en P_H para ser buscado dentro del conjunto de señales de ECG digitales almacenados.

En el caso de las aplicaciones ad-hoc, cada conjunto de datos fue almacenado en directorios distintos, cada uno identificado por el nombre de la base de datos utilizada. Cada una de las señales de ECG digital fue almacenada en un archivo independiente con un formato binario. De igual manera, para el almacenamiento de los patrones se usó un directorio por cada conjunto de datos, y cada patrón fue almacenado dentro de un archivo independiente con formato binario. El identificador tanto para las señales como para los patrones fue asignado como el nombre del archivo en el cual fueron almacenados.

4.3.7. Tiempos de Ejecución Para el Algoritmo K-NN

Para llevar a cabo la experimentación utilizando las consultas propuestas y la aplicación K_ADHOC para resolver K-NN fue necesario establecer un valor para la variable K. Se definieron tres valores: $K = 1$, $K = 10$ y $K = 100$. Así, a los algoritmos que resuelven la problemática K-NN los identificamos como 1-NN, 10-NN y 100-NN. De igual manera, para la aplicación K_ADHOC se utilizaron los tres valores para K con el fin de tener un punto de comparación. De este modo, la aplicación K_ADHOC se identifica como K1_ADHOC, K10_ADHOC y K100_ADHOC para los tres valores de K. Para cualquier valor de K, los tiempos obtenidos por la aplicación K_ADHOC representan para nuestra propuesta los tiempos de ejecución a igualar y/o mejorar.

En la Tabla 4.4 se observan los tiempos de ejecución (medidos en segundos), obtenidos por el algoritmo 1-NN para las 15 consultas propuestas, así como para la aplicación K1_ADHOC sobre los cuatro conjuntos de datos contemplados. En ella, se resaltan los mejores tiempos de ejecución para cada base de datos. Para la base de datos AAMI, se observan los menores tiempos de ejecución debido al número de señales que contiene así como su duración. Por otro lado, la base de datos European presenta los mayores tiempos de ejecución al contener las señales con mayor duración. Para un mismo conjunto de datos, se observan tiempos de ejecución que difieren en

Consulta	Base de Datos			
	AAMI	QT	MIT-BIH	European
H_TMP_K_V1	1.687	50.945	88.679	380.182
H_TMP_K_V2	1.699	61.912	104.479	470.472
H_TMP_K_V3	2.956	84.711	269.522	1,504.732
H_TMP_K_V4	1.560	50.542	90.145	385.893
H_VIEW_K_V1	1.647	40.601	76.808	312.403
H_VIEW_K_V2	1.453	52.408	93.026	407.199
H_VIEW_K_V3	2.117	115.636	174.940	935.966
H_VIEW_K_V4	2.439	80.105	151.864	616.614
H_WITH_K_V1	1.294	40.908	76.403	317.337
H_WITH_K_V2	1.474	52.911	92.553	413.155
H_WITH_K_V3	2.239	133.776	194.470	1,049.440
H_WITH_K_V4	1.308	41.519	77.349	323.889
V_TMP_K	7.581	891.798	1,536.092	11,608.204
V_VIEW_K	5.712	596.316	1,000.104	5,823.180
V_WITH_K	6.145	786.600	1,261.448	10,080.957
K1_ADHOC	1.314	39.819	71.995	298.163

Tabla 4.4: Tiempos de Ejecución en Segundos Para el Algoritmo 1-NN

gran medida, por el modelo de almacenamiento utilizado. Para las consultas sobre el modelo de almacenamiento horizontal en la base de datos AAMI se obtuvieron tiempos promedio de 1.823 segundos, mientras que para el modelo de almacenamiento vertical, se obtuvieron tiempos promedio de 6.48 segundos. Esto representa una diferencia de 3.55 veces más tiempo para el modelo de almacenamiento vertical. Este comportamiento se sigue reflejando para las otras tres bases de datos, donde los tiempos de ejecución promedio sobre el modelo almacenamiento vertical fueron mayores en una proporción de 11.3 veces, con respecto al modelo de almacenamiento horizontal para la base de datos QT. En el caso de la base de datos MIT-BIH se obtuvieron 12.2 veces más tiempo, mientras que para la base de datos European se obtuvieron 15.46 veces más tiempo de ejecución. De las 12 consultas propuestas para el modelo de almacenamiento horizontal, aquellas que obtuvieron el menor tiempo de ejecución para el algoritmo 1-NN fueron H_WITH_K_V1 y H_VIEW_K_V1 con rangos de 0.36, 0.41, 0.31 y 4.9 segundos para las bases de datos AAMI, QT, MIT-BIH y European, respectivamente. Para cada una de las bases de datos las consultas que

Consulta	Base de Datos			
	AAMI	QT	MIT-BIH	European
H_TMP_K_V1	1.630	50.364	88.167	377.089
H_TMP_K_V2	1.691	50.063	104.619	470.662
H_TMP_K_V3	2.863	85.103	268.301	1,501.662
H_TMP_K_V4	1.611	51.452	92.038	397.054
H_VIEW_K_V1	2.175	42.122	83.629	327.870
H_VIEW_K_V2	2.299	54.613	99.989	418.473
H_VIEW_K_V3	2.020	115.515	173.569	936.840
H_VIEW_K_V4	13.226	423.835	827.093	3,339.562
H_WITH_K_V1	1.310	40.842	76.054	317.212
H_WITH_K_V2	1.491	52.874	92.509	413.189
H_WITH_K_V3	2.188	133.387	194.515	1,053.010
H_WITH_K_V4	1.288	41.113	77.452	323.937
V_TMP_K	7.337	897.749	1,584.889	12,809.372
V_VIEW_K	5.803	588.740	1,003.811	6,123.465
V_WITH_K	6.769	772.303	1,243.804	10,462.336
K10_ADHOC	1.318	39.995	71.362	297.889

Tabla 4.5: Tiempos de Ejecución en Segundos para el Algoritmo 10-NN

más se acercaron a los tiempos obtenidos por la aplicación K1_ADHOC fueron las siguientes: la consulta H_WITH_K_V1 obtuvo el mejor tiempo de ejecución para la base de datos AAMI, mejorando al tiempo requerido por la aplicación K1_ADHOC necesitando sólo el 98.5 % del tiempo de ejecución de esta última . Para la base de datos QT la consulta con mejor tiempo de ejecución fue H_VIEW_K_V1, la cual obtuvo un 101.9 % del tiempo de ejecución necesario por la aplicación K1_ADHOC. La consulta H_WITH_K_V1 fue la mejor para la base de datos MIT-BIH, con tiempo de ejecución equivalente al 106.1 % de los obtenidos por la aplicación K1_ADHOC. Finalmente, para la base de datos European, la consulta con menor tiempo de ejecución fue H_VIEW_K_V1, equivalente al 104.7 % del tiempo de ejecución obtenido por la aplicación K1_ADHOC.

Los tiempos de ejecución para el algoritmo 10-NN, se observan en la Tabla 4.5. En ella, se resaltan los mejores tiempos de ejecución para cada base de datos. Al incrementar el valor de K a 10, la mayoría de las consultas así como la aplicación K10_ADHOC presentan un tiempo de ejecución constante respecto al obtenido por

el algoritmo K1_ADHOC. Sin embargo, se obtuvo un incremento significativo sobre el tiempo de ejecución de las consultas H_TMP_V4 y H_VIEW_V4 principalmente. La primera de ellas presenta una diferencia total para las cuatro bases de datos de 14 segundos, lo que representa un incremento promedio del 2.6 % con respecto al algoritmo 1-NN. Por otro lado, la consulta H_VIEW_V4 obtuvo un incremento de 3,752.7 segundos, lo que representa un aumento del 440.9 % con respecto al tiempo de ejecución obtenido por el algoritmo 1-NN. Esto último es el comportamiento más notorio entre los algoritmos 1-NN y 10-NN para todas las consultas. Dicho incremento en el tiempo de ejecución se atribuye a la múltiple ejecución de las vistas T1 y T3 presentes en la cuarta subconsulta de H_VIEW_V4 al momento de utilizar la búsqueda de valores por índice sobre arreglos. Para el algoritmo 10-NN, las consultas sobre el modelo de almacenamiento vertical, presentan tiempos de ejecución superiores a los obtenidos por las consultas sobre el modelo de almacenamiento horizontal para las cuatro bases de datos. Para la base de datos AAMI, se presentaron tiempos mayores en una proporción de 2.3 más en el modelo de almacenamiento vertical que con respecto al modelo de almacenamiento horizontal. Por otro lado, se aprecian tiempos de ejecución 7.9 veces superiores para la base de datos QT, 6.9 veces más para la base de datos MIT-BIH y 11.1 veces más para la base de datos European. Las consultas que más se acercaron al tiempo de ejecución obtenido por la aplicación K10_ADHOC, fueron las siguientes: para la base de datos AAMI, la consulta con menor tiempo de ejecución fue H_WITH_V4, mejorando el tiempo de ejecución de la aplicación K10_ADHOC al requerir solamente el 96.6 % del tiempo necesario por esta última. Para la base de datos QT, la mejor consulta fue H_WITH_V1, con un tiempo de ejecución que representa un 102 % al tiempo de ejecución obtenido por la aplicación K10_ADHOC. La consulta H_WITH_V1 representa también la mejor consulta para la base de datos MIT-BIH con un tiempo de ejecución equivalente al 105.6 % del requerido por la aplicación K10_ADHOC. Finalmente, para la base de datos European la consulta con menor tiempo de ejecución fue también H_WITH_V1, con tiempo de

Consulta	Base de Datos			
	AAMI	QT	MIT-BIH	European
H_TMP_K_V1	1.689	50.473	88.991	377.141
H_TMP_K_V2	1.676	61.789	105.175	469.519
H_TMP_K_V3	2.872	85.244	270.002	1,502.144
H_TMP_K_V4	2.167	63.031	115.684	533.252
H_VIEW_K_V1	10.184	59.297	154.173	480.462
H_VIEW_K_V2	10.324	71.292	170.051	571.319
H_VIEW_K_V3	2.041	115.969	174.254	937.894
H_VIEW_K_V4	121.108	3,960.133	7,577.728	30,559.951
H_WITH_K_V1	1.367	40.891	78.044	321.029
H_WITH_K_V2	1.498	53.207	93.973	413.714
H_WITH_K_V3	2.196	133.310	194.523	1,052.587
H_WITH_K_V4	1.277	41.320	77.448	324.237
V_TMP_K	7.940	888.050	1,602.689	12,943.251
V_VIEW_K	5.955	605.025	999.568	6,062.167
V_WITH_K	6.751	769.424	1,246.571	10,959.465
K100_ADHOC	1.322	40.099	72.017	298.491

Tabla 4.6: Tiempos de Ejecución en Segundos para el Algoritmo 100-NN

ejecución equivalente al 106.5 % del tiempo de ejecución obtenido por la aplicación K10_ADHOC.

Para el algoritmo 100-NN se obtuvieron los tiempos de ejecución observados en la tabla 4.6. En ella, se resaltan los mejores tiempos de ejecución para cada base de datos. Al establecer el valor de $K = 100$ se presenta el mismo comportamiento observado por el algoritmo 10-NN. Es decir, se observan tiempos de ejecución constantes para la mayoría de las consultas con respecto a los algoritmos 1-NN y 10-NN. No obstante, las consultas H_TMP_V4, H_VIEW_V2 y H_VIEW_V4 obtuvieron un incremento significativo en el tiempo de ejecución promedio para las cuatro bases de datos. Con respecto al algoritmo 10-NN, la consulta H_TMP_V4 obtuvo un incremento de 172 segundos, lo que representa un aumento promedio del 31.7 %. Por otro lado, las consultas H_VIEW_V2 y H_VIEW_V4 obtuvieron un incremento en los tiempos de ejecución del 43 % y 817 % respectivamente. A diferencia de los algoritmos 1-NN y 10-NN, los tiempos promedio para cada base de datos en el algoritmo 100-NN fueron inferiores para el modelo de almacenamiento vertical. Sin embargo, esto se ve afectado de manera significativa por los tiempos de ejecución de la consulta H_VIEW_V4, la

cual para las bases de datos AAMI, QT, MIT-BIH y European representa el 76.5 %, 72.2 %, 90.7 % y 81.4 % respectivamente sobre los tiempos totales de ejecución por base de datos. Al omitir los tiempos de ejecución de la consulta H_VIEW_V4 que sesgan por completo los datos, las consultas sobre el modelo de almacenamiento horizontal presentan al igual que en 1-NN y 10-NN tiempos inferiores a los obtenidos por las consultas sobre el modelo de almacenamiento vertical. Las consultas sobre el modelo de almacenamiento horizontal para la base de datos AAMI obtuvieron 2.2 veces menos tiempo promedio de ejecución en comparación con las consultas sobre el modelo de almacenamiento vertical. Esto mismo para la base de datos QT donde las consultas sobre el modelo de almacenamiento horizontal obtuvieron tiempos 9.7 veces menores. Para la base de datos MIT-BIH fueron 10.7 veces menos tiempo, mientras que para la base de datos European los tiempos de ejecución fueron menores en una proporción de 15.7 veces.

4.3.8. Tiempos de Ejecución Para el Algoritmo ϵ-search

Llevar a cabo la experimentación utilizando las consultas propuestas así como la aplicación E_ADHOC para resolver la problemática ϵ-search fue necesario definir el valor para la variable ϵ. Con el fin de tener un punto de comparación sobre la experimentación de la problemática K-NN, el valor de ϵ fue definido de tal manera que para cada patrón almacenado dada una base de datos, el número de resultados en la salida final sea idéntica a la obtenida en K-NN. Esto es, para el algoritmo 1-NN se definió un algoritmo denominado E1-Search. En el algoritmo E1-Search, al valor de ϵ para buscar el patrón q_j dada una base de datos, le fue asignado el primer y único valor de semejanza obtenido por el algoritmo 1-NN al buscar el patrón q_j sobre la misma base de datos. De igual manera, se definió un algoritmo E2-Search en donde al valor de ϵ se le asignó el décimo valor de semejanza obtenido por el algoritmo 10-NN al buscar un mismo patrón para una misma base de datos. Finalmente, el algoritmo

Consulta	Base de Datos			
	AAMI	QT	MIT-BIH	European
H_TMP_E_V1	1.652	50.108	88.365	377.598
H_TMP_E_V2	2.139	112.729	181.184	900.038
H_TMP_E_V3	3.277	260.220	361.245	2,126.666
H_TMP_E_V4	1.892	57.452	101.510	402.502
H_VIEW_E_V1	1.357	41.509	76.007	315.894
H_VIEW_E_V2	1.525	48.847	87.577	378.877
H_VIEW_E_V3	1.938	110.728	161.538	896.852
H_VIEW_E_V4	2.795	92.875	162.916	649.498
H_WITH_E_V1	1.372	41.658	76.711	324.600
H_WITH_E_V2	1.670	68.631	113.066	536.284
H_WITH_E_V3	2.468	155.111	220.615	1,221.206
H_WITH_E_V4	1.492	48.772	84.193	352.894
V_TMP_E	7.791	945.045	1,530.912	12,430.381
V_VIEW_E	6.889	917.313	1,193.682	10,841.375
V_WITH_E	7.070	845.709	1,231.767	11,854.029
E1_ADHOC	1.461	40.816	72.618	299.507

Tabla 4.7: Tiempos de Ejecución en Segundos para el Algoritmo E1-Search

E3-Search fue definido utilizando el centésimo valor obtenido por el algoritmo 100-NN al buscar un mismo patrón sobre una misma base de datos.

De este modo, los tiempos de ejecución obtenidos por el algoritmo E1-Search se observan en la Tabla 4.7. En ella, se resaltan los mejores tiempos de ejecución para cada base de datos. En la Tabla 4.7, se identifica un comportamiento similar al de la Tabla 4.4 para el algoritmo 1-NN. Este comportamiento reside en la diferencia de tiempo de ejecución entre las consultas sobre el modelo de almacenamiento horizontal y las consultas sobre el modelo de almacenamiento vertical. Para las bases de datos AAMI, QT, MIT-BIH y European, los tiempos de ejecución de las consultas sobre el modelo de almacenamiento vertical fueron superiores en una proporción de 3.7, 9.9, 9.2 y 16.6 veces respectivamente. Los tiempos de ejecución de las consultas utilizadas para el algoritmo E1-Search sobre el modelo de almacenamiento horizontal se ven directamente afectados por el ordenamiento de datos que requiere, por medio de la cláusula ORDER BY. Sin embargo, esto resulta indiferente para las consultas con Versión V1, es decir, H_TMP_E_V1, H_VIEW_E_V1 y H_WITH_E_V1, las cuales

presentan tiempos muy similares a las consultas equivalentes (aquellas consultas sobre un mismo modelo de almacenamiento, utilizando un mismo medio para el tratamiento de datos intermedios y misma versión) para el algoritmo 1-NN. Esto se debe a que la segunda subconsulta realiza una selección de solamente un valor de semejanza, por lo que el ordenamiento representa una tarea que requiere de pocos recursos. Esto mismo sucede en las consultas con Versión V4, donde los tiempos de ejecución con respecto a las consultas equivalentes con el algoritmo 1-NN son superiores con un promedio de 45.2 segundos, lo que representa un incremento de apenas un 6.9 % en el tiempo de ejecución para las cuatro bases de datos. Por otro lado, para las consultas con Versiones V2 y V3, los tiempos de ejecución con respecto a los obtenidos por las consultas equivalentes en el algoritmo 1-NN, presentan valores superiores de manera significativa. Esto debido al ordenamiento realizado en la segunda subconsulta sobre el número total de valores de semejanza calculados. Las consultas con Versión V2 para el algoritmo E1-Search presentan tiempos superiores en un 27.9 % con respecto a las consultas equivalentes del algoritmo 1-NN. Para las consultas con Versión V3, los tiempos de ejecución resultan superiores en un 20 % con respecto a las consultas equivalentes del algoritmo 1-NN. La consulta con mejores tiempos de ejecución para el algoritmo E1-Search fue H_VIEW_E_V1 en las cuatro bases de datos. Para la base de datos AAMI obtuvo un tiempo de ejecución de 1.357 segundos, mejorando al obtenido por la aplicación E1_ADHOC, la cual requirió de 1.461 segundos. Esto representa una mejoría de un 7.1 % por parte de la consulta H_VIEW_E_V1. En el caso de la base de datos QT, se obtuvo un tiempo de ejecución equivalente al 101.7 % del tiempo de ejecución obtenido por la aplicación E1_ADHOC. Por otra parte, para la base de datos MIT-BIH se obtuvo un tiempo de ejecución equivalente al 104.7 % del tiempo de ejecución que le tomó a la aplicación E1_ADHOC. Finalmente, para la base de datos European se obtuvo un tiempo de ejecución que representa el 105.5 % del requerido por la aplicación E1_ADHOC.

En la Tabla 4.8 se observan los tiempos de ejecución obtenidos por el algoritmo

Consulta	Base de Datos			
	AAMI	QT	MIT-BIH	European
H_TMP_E_V1	1.701	50.445	88.578	378.951
H_TMP_E_V2	2.306	114.818	177.761	906.170
H_TMP_E_V3	3.353	259.271	364.503	2,156.639
H_TMP_E_V4	2.320	59.582	102.903	409.870
H_VIEW_E_V1	2.191	42.204	83.033	330.144
H_VIEW_E_V2	2.291	50.689	94.602	394.153
H_VIEW_E_V3	1.927	110.392	160.650	897.245
H_VIEW_E_V4	15.918	519.920	940.732	3,532.481
H_WITH_E_V1	1.363	41.058	76.648	323.398
H_WITH_E_V2	1.587	68.362	116.490	541.343
H_WITH_E_V3	2.373	153.686	219.434	1,235.202
H_WITH_E_V4	1.396	49.102	84.254	353.209
V_TMP_E	7.508	949.845	1,599.187	12,484.130
V_VIEW_E	7.245	937.052	1,245.353	11,272.655
V_WITH_E	7.113	842.492	1,267.120	11,833.330
E2_ADHOC	1.491	40.907	72.680	298.464

Tabla 4.8: Tiempos de Ejecución en Segundos para el Algoritmo E2-Search

E2-Search. En ella, se resaltan los mejores tiempos de ejecución para cada base de datos. Al igual que en el algoritmo 10-NN, el comportamiento más notorio es el aumento significativo en el tiempo de ejecución para la consulta H_VIEW_K_V4 en las cuatro bases de datos. Para las bases de datos AAMI, QT, MIT-BIH y European los tiempos de ejecución de la consulta H_VIEW_E_V4 respecto a los obtenidos en el algoritmo E1-Search son superiores en una proporción de 5.7, 5.8, 5.6 y 5.4 veces respectivamente. Los tiempos promedio de las consultas sobre el modelo horizontal resultan superiores en términos de eficiencia contra las consultas sobre el modelo vertical. Esta diferencia resulta en promedio 2.2, 6.3, 7.1 y 12.3 veces menos tiempo por parte de las consultas sobre el modelo de almacenamiento horizontal para las bases de datos AAMI, QT, MIT-BIH y European respectivamente. La mejor consulta para el algoritmo E2-Search fue H_WITH_E_V1 en todas las bases de datos. A diferencia del algoritmo E1-Search, donde la mejor consulta fue H_VIEW_E_V1, en el algoritmo E2-Search los tiempos de ejecución para H_VIEW_E_V1 aumentaron al incrementar el umbral de semejanza ϵ para que el número de resultados devueltas sea igual a

Consulta	Base de Datos			
	AAMI	QT	MIT-BIH	European
H_TMP_E_V1	1.848	50.724	88.474	376.951
H_TMP_E_V2	2.330	113.506	178.933	933.209
H_TMP_E_V3	3.361	262.668	359.704	2,156.639
H_TMP_E_V4	4.339	68.920	122.389	716.916
H_VIEW_E_V1	10.276	59.249	153.340	330.144
H_VIEW_E_V2	10.301	67.698	164.931	394.153
H_VIEW_E_V3	1.933	110.973	160.938	897.245
H_VIEW_E_V4	142.115	4,380.720	8,320.581	33,928.154
H_WITH_E_V1	1.397	41.243	77.748	323.398
H_WITH_E_V2	1.611	68.276	115.565	541.343
H_WITH_E_V3	2.418	155.719	219.784	1,235.202
H_WITH_E_V4	1.426	49.190	84.272	354.915
V_TMP_E	7.549	963.840	1,688.354	12,421.440
V_VIEW_E	7.665	960.907	1,290.040	11,298.836
V_WITH_E	8.856	842.745	1,291.965	11,921.125
E3_ADHOC	1.575	41.117	73.648	306.437

Tabla 4.9: Tiempos de Ejecución en Segundos para el Algoritmo E3-Search

10. Para la base de datos AAMI, la consulta H_WITH_E_V1 mejoró los tiempos de ejecución obtenidos por la aplicación E2_ADHOC, al requerir solamente el 91.4 % del tiempo de ejecución de esta última. Para las bases de datos QT, MIT-BIH y European, la consulta H_WITH_E_V1 presentó tiempos de ejecución equivalentes al 100.4 %, 105.5 % y 108.4 % respectivamente contra los tiempos de ejecución de la aplicación E2_ADHOC.

Finalmente, para el algoritmo E3-Search se obtuvieron los tiempos de ejecución observados en la Tabla 4.9. En ella, se resaltan los mejores tiempos de ejecución para cada base de datos. En esta tabla se observa un incremento notable en los tiempos de ejecución para la consulta H_VIEW_E_V4, tal y como se observó también en el algoritmo 100-NN. Con respecto a los tiempos de ejecución del algoritmo E2-Search, este incremento se aproxima en una proporción de nueve veces para las cuatro bases de datos. Para la base de datos AAMI se tuvo un incremento de 8.9 veces. Por otro lado, para la base de datos QT se obtuvo un incremento de 8.8 veces. De igual manera, para las bases de datos MIT-BIH y European se obtuvo un incremento de 8.4 y 9.6 veces respectivamente. Durante la ejecución de los algoritmos E1-Search, E2-Search

y E3-Search se obtuvieron de manera constante los mayores tiempos de ejecución en las consultas sobre el modelo horizontal H_TMP_E_V3 y H_WITH_E_V3. Este comportamiento se atribuye a la tercer subconsulta, la cual hace uso de una función de ventana para la búsqueda de la posición donde aparece un patrón determinado. Esto a su vez puede verse relacionado con las consultas sobre el modelo de almacenamiento vertical, donde de igual manera se utilizan funciones de ventana, con lo que se observan los mayores tiempos de ejecución para todos los algoritmos contemplados durante nuestra etapa de experimentación. Una vez más, la consulta H_WITH_E_V1 obtuvo el mejor tiempo de ejecución para las cuatro bases de datos. Los tiempos de ejecución sobre la base de datos AAMI de la aplicación E3_ADHOC fueron mejorados al requerir solamente el 88.7 % del tiempo de ejecución de esta última. Para las otras tres bases de datos se obtuvieron tiempos de ejecución muy cercanos a los obtenidos por la aplicación E3_ADHOC. El tiempo de ejecución de la consulta H_WITH_E_V1 para la base de datos QT, obtuvo un tiempo de ejecución equivalente al 100.3 % del requerido por la aplicación E3_ADHOC. Para la base de datos MIT-BIH el tiempo de ejecución obtenido equivale al 105.6 % del requerido por la aplicación E3_ADHOC. Finalmente, para la base de datos European, el tiempo de ejecución para la consulta H_WITH_E_V1 fue equivalente al 105.5 % del obtenido por la aplicación E3_ADHOC.

4.3.9. Tiempos de Ejecución Sobre una Base de Datos de Gran Tamaño

Observando los resultados de los algoritmos que resuelven las problemáticas K-NN y ϵ-Search, se observan comportamientos muy similares entre las consultas equivalentes. Sin embargo, en términos de tiempo de ejecución, los algoritmos para resolver K-NN resultan ser más eficientes. Dentro de los algoritmos 1-NN, 10-NN y 100-NN, las consultas sobre el modelo de almacenamiento horizontal fueron indudablemente más eficientes que aquellos obtenidos por las consultas sobre el modelo de almacenamiento vertical. Dentro de las consultas sobre el modelo de almacenamiento horizontal, aque-

llas que utilizan CTEs para el tratamiento de datos intermedios fueron, en términos generales, las que obtuvieron los mejores tiempos de ejecución. En segundo lugar, las consultas que utilizan tablas transitorias, con una eficiencia que equivale al 127.9 % de los tiempos obtenidos por las consultas que utilizan CTEs. Por último, las consultas que utilizan vistas representan en promedio, los tiempos de ejecución menos eficientes, siendo 548 % más lentas que aquellas que utilizan CTEs.

Estas circumstancias se dieron sobre bases de datos cuyo requerimiento en disco es menor a la memoria disponible en el sistema, siendo la base de datos European la de mayor tamaño (417MB). Sin embargo, contemplando un escenario en el cuál el tamaño total del conjunto de datos almacenados sea mayor a la memoria principal disponible por el sistema, se realizó la experimentación sobre un conjunto de datos que cuente con dicha característica. Esto con el fin de analizar el comportamiento de las distintas opciones para el tratamiento de datos intermedios bajo este escenario. Para ello, se prodeció a crear una nueva base de datos, a la cual denominamos GranDB. La base de datos GranDB fue almacenada en una tabla *h_big* bajo el esquema del modelo de almacenamiento horizontal. En ella, se almacenaron los datos del conjunto de prueba European. Sin embargo, cada una de las señales presentes en la base de datos European fue almacenada cuatro veces. Con ello, el tamaño total requerido fue de 1670MB, lo que representa más que la memoria disponible por el sistema, la cual oscilaba entre los 1,200MB y 1,300MB sin carga de trabajo. De manera similar, se creó una base de datos GranDB_Bin para almacenar cuatro veces los datos presentes en archivos binarios correspondientes a la base de datos European. Esto con el fin de implementar las aplicaciones K1_ADHOC, K10_ADHOC y K100_ADHOC. De esta forma, los datos binarios presentaron un espacio de almacenamiento total de 5,040 MB. Las consultas evaluadas fueron aquellas que presentaron los mejores tiempos de ejecución. Es decir, las consultas sobre el modelo de almacenamiento horizontal, para el algoritmo K-NN con los valores de $K = 1$, $K = 10$ y $K = 100$ con el fin de observar su comportamiento. Los patrones utilizados para ello, fueron reutilizados de la tabla

Consulta	Algoritmo		
	1-NN	10-NN	100-NN
H_TMP_K_V1	1,507.6	1,509.6	1,514.4
H_TMP_K_V2	1,870.4	1,869.6	1,875.0
H_TMP_K_V3	3,764.6	3,756.8	3,757.8
H_TMP_K_V4	1,512.9	1,586.6	2,133.4
H_VIEW_K_V1	1,243.3	1,258.6	1,744.1
H_VIEW_K_V2	1,636.9	1,621.9	1,777.3
H_VIEW_K_V3	4,497.7	4,506.4	4,518.1
H_VIEW_K_V4	2,468.7	13,865.1	122,085.8
H_WITH_K_V1	1,493.6	1,496.3	1,507.6
H_WITH_K_V2	1,644.0	1,642.5	1,644.7
H_WITH_K_V3	4,862.6	4,903.8	4,915.7
H_WITH_K_V4	1,492.7	1,488.2	1,496.3
K_ADHOC	1,182.1	1,184.7	1,184.5

Tabla 4.10: Tiempos de Ejecución en Segundos para el Algoritmo K-NN y el Modelo de Almacenamiento Horizontal en una Base de Datos de Gran Tamaño

ya creada *h_edb_pat*.

En la Tabla 4.10 se observan los tiempos de ejecución obtenidos para los algoritmos 1-NN, 10-NN y 100-NN sobre las bases de datos GranDB y GranDB_Bin, utilizando las consultas sobre el modelo de almacenamiento horizontal y las aplicaciones ADHOC. En ella se hacen notorios los tiempos de ejecución de la consulta H_VIEW_K_V4, los cuales se encuentran muy por encima del resto de los tiempos observados por las otras consultas. Sin embargo, este mismo comportamiento fue notado sobre los experimentos en las bases de datos AAMI, QT, MIT-BIH y European, tanto para los algoritmos que resuelven K-NN como para aquellos que resuelven ϵ-search. Al contener cuatro veces los datos de la base de datos European, se espera que los tiempos de ejecución sobre las bases de datos GranDB y GranDB_Bin conserven esta correlación y obtengan tiempos de ejecución superiores en una proporción de cuatro veces con respecto a los obtenidos sobre la base de datos European. Sin embargo, en la tabla 4.11 podemos observar las proporciones en el incremento de los tiempos de ejecución sobre las bases de datos GranDB y GranDB_Bin en comparación a los obtenidos sobre la base de datos European. Dicha comparación abarca todas las consultas

Consulta	Algoritmo		
	1-NN	10-NN	100-NN
H_TMP_K_V1	3.97	4.00	4.02
H_TMP_K_V2	3.98	3.97	3.99
H_TMP_K_V3	2.50	2.50	2.50
H_TMP_K_V4	3.92	4.00	4.00
H_VIEW_K_V1	3.98	3.84	3.63
H_VIEW_K_V2	4.02	3.88	3.11
H_VIEW_K_V3	4.80	4.81	4.82
H_VIEW_K_V4	4.00	4.15	3.99
H_WITH_K_V1	4.71	4.72	4.70
H_WITH_K_V2	3.98	3.98	3.98
H_WITH_K_V3	4.63	4.66	4.67
H_WITH_K_V4	4.61	4.59	4.61
C_ADHOC	3.96	3.98	3.97

Tabla 4.11: Proporciones de Incremento en Tiempos de Ejecución en la Base de Datos GranDB Respecto a los Tiempos de Ejecución Sobre la Base de Datos European

sobre el modelo de almacenamiento horizontal y las aplicaciones ADHOC para los algoritmos 1-NN, 10-NN y 100-NN. En ella, podemos observar proporciones mayores a cuatro, lo que indica que no existe correlación en los tiempos de ejecución al incrementar el tamaño de la base de datos que se utiliza. Esto se presenta principalmente en las alternativas de datos intermedios que se especializan en el procesamiento de datos en memoria, debido al número de datos que requieren ser leídos y escritos en disco. Las aplicaciones ADHOC sobre archivos binarios presentan la correlación en tiempos de ejecución al incrementar el número de datos a procesar. Por otro lado, la mejor alternativa al incrementar el número de datos son las tablas transitorias, presentanto un incremento promedio en los tiempos de ejecución de 2.5 veces más al incrementar los datos en una proporción de cuatro veces. La razón de esto es la especialización que presentan para trabajar con datos escritos y leídos en disco, lo que se hace extrictamente necesario al tener una base de datos cuyo tamaño es mayor a la memoria disponible por el sistema.

De esta manera, damos paso a las observaciones finales de nuestro trabajo, así como las conclusiones que se encontraron con ayuda de la experimentación realizada. Por último, presentamos nuestros ideales para trabajo futuro. Con esto, se espera mejorar la propuesta que hacemos en este trabajo de tesis, tanto en términos de tiempo de ejecución, como en términos de espacio en disco requerido por los datos con el uso de nuevas técnicas y tecnologías.

Capítulo 5
Conclusiones

En este trabajo de tesis hemos propuesto un marco de trabajo para lograr la implementación del procesamiento de señales de electrocardiograma (ECG) digitales dentro de un sistema gestor de bases de datos (SGBD), sin necesidad de utilizar aplicaciones externas. La propuesta implementa un conjunto de operadores básicos, que pueden ser reutilizados en distintos algoritmos, haciendo uso de funciones definidas por el usuario (UDFs) ofrecidas por los SGBD comerciales actuales. Además, se proponen dos modelos de almacenamiento de datos para representar a las señales de ECG digitales dentro de un SGBD. A estos modelos de almacenamiento los denominamos modelo de almacenamiento horizontal y modelo de almacenamiento vertical, respectivamente. Para cada uno de estos modelos de almacenamiento, se propone la estructura general de las consultas necesarias para realizar las operaciones contempladas dentro del campo de búsqueda por semejanza sobre señales de ECG digitales. Dentro de estas consultas, se tomaron en cuenta distintos medios para el tratamiento de datos intermedios entre subconsultas, descritos en la Sección 4.3.1.

Nuestra implementación de algoritmos de búsqueda por semejanza sobre señales de ECG digitales almacenadas en un SGBD fue desarrollada bajo el enfoque de la búsqueda de subsecuencias, sobre las problemáticas de búsqueda por vecinos más cercanos (K-NN) y búsqueda por rango (ϵ-search). Dado que la búsqueda por subsecuencias no es más que una generalización de la búsqueda completa, podemos afirmar que se abarca por completo la implementación de búsqueda por semejeanza sobre señales de ECG digitales dentro de un SGBD. Aunado a ello, nos fue posible obtener tiempos de ejecución por parte de nuestra propuesta que superan en términos de eficiencia a aquellos tiempos de ejecución obtenidos por parte aplicaciones ad-hoc.

Como consecuencia, queda demostrado que es posible implementar operadores básicos que conformen los algoritmos para realizar búsqueda por semejanza sobre señales de ECG digitales como funciones propias de un SGBD, sin necesidad de utilizar herramientas externas. Esto último representa la validación de la hipótesis estipulada en este trabajo de tesis.

Los resultados obtenidos por los modelos de almacenamiento propuestos en este trabajo, reflejan una ventaja significativa por parte del modelo de almacenamiento horizontal sobre el modelo de almacenamiento vertical, para todos los algoritmos contemplados. De todas las consultas evaluadas para los algoritmos 1-NN, 10-NN y 100NN, las consultas sobre el modelo de almacenamiento horizontal resultaron ser más eficientes de manera definitiva. En general, las consultas sobre el modelo de almacenamiento horizontal resultaron ser 41.2 % más eficientes que las consultas sobre el modelo de almacenamiento vertical. Esto mismo sucedió con aquellas consultas para los algoritmos que resuelven la problemática ϵ-search. Es decir, para los algoritmos E1-Search, E2-Search y E3-Search, las consultas sobre el modelo de almacenamiento horizontal resultaron ser más eficientes que aquellas sobre el modelo de almacenamiento vertical en un 50.7 %.

De manera definitiva, podemos concluír que la mejor opción es utilizar el modelo de almacenamiento horizontal para almacenar señales de ECG dentro de un SGBD. Esto, en primer lugar por requerir un menor espacio en disco, aproximadamente 3 veces menos que el necesario por archivos binarios y 34 veces menos que el espacio en disco requerido por el modelo de almacenamiento vertical. En segundo lugar, porque bajo el modelo de almacenamiento horizontal se obtuvieron los tiempos de ejecución más cercanos, e incluso mejorando en algunos casos a los obtenidos por aplicaciones ad-hoc. Cuando el tamaño total de los datos no supera la memoria disponible por el sistema, podemos concluír que la mejor opción es utilizar CTEs, debido a que estas se especializan en la creación y procesamiento de tablas en memoria, y fue de manera constante la opción que obtuvo los mejores tiempos de ejecución sobre todos

los algoritmos. Esto manteniendo tiempos constantes para los algoritmos sobre una misma problemática, sin importar la variación del valor de K para la problemática K-NN, o el valor de ϵ para la problemática ϵ-search. Por otro lado, cuando el tamaño total de los datos a procesar supere la memoria disponible por el sistema, la mejor opción resultan ser las tablas transitorias. Esto último, gracias a la propiedad nativa de las tablas transitorias para la escritura/lectura de datos directo en disco, y fué demostrado en la Sección 4.3.9.

Con el surgimiento de nuevas tecnologías, se desea probar nuevas opciones tanto en SGBD como en nuevos modelos de almacenamiento para las señales de ECG digitales. Todo esto con la finalidad de reducir más los tiempos de ejecución obtenidos en nuestro trabajo, manteniendo las bondades que nos ofrecen los SGBD actuales. Junto a esto, la utilización de técnicas de indexación y reducción en la dimensionalidad de las señales de ECG digitales representa una posibilidad para obtener tiempos de ejecución aún mejores. Por lo que queda contemplado como parte de nuestra investigación y trabajos futuros.

Bibliografía

[1] Daniel Abadi and Samuel Madden. Query execution in column-oriented database systems. Technical report, 2008.

[2] Daniel Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: How different are they really? In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 967–980, New York, NY, USA, 2008. ACM.

[3] Rejendra Acharya, Suri Jasjit, Jos Spaan, Shankar Krishnan, and Biocom Technologies. *Advances in Cardiac Signal Processing*. Springer Berlin Heidelberg, 2007.

[4] Valtino Afonso, Willis Tompkins, Truong Nguyen, Kurt Michler, and Shen Luo. Comparing stress ecg enhancement algorithms. *Engineering in Medicine and Biology Magazine, IEEE*, 15(3):37–44, May 1996.

[5] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. *Efficient similarity search in sequence databases*. FODO '93. Springer-Verlag, London, UK, UK, 1993.

[6] Paul Albrecht. *ST Segment Characterization for Long Term Automated ECG Analysis*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1983.

[7] Krisztian Buza, Alexandros Nanopoulos, and Lars Schmidt. Fusion of similarity measures for time series classification. In *Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems - Volume Part II*, HAIS'11, pages 253–261. Springer-Verlag, Berlin, Heidelberg, 2011.

[8] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. isax 2.0: Indexing and mining one billion time series. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 58–67, Dec 2010.

[9] Qiming Chen, Meichun Hsu, Riu Liu, and Weihong Wang. Scaling-up and speeding-up video analytics inside database engine. In Sourav S. Bhowmick, Josef Kang, and Roland Wagner, editors, *Database and Expert Systems Applications, 20th International Conference, DEXA 2009, Linz, Austria, August 31 - September 4, 2009. Proceedings*, volume 5690 of *Lecture Notes in Computer Science*, pages 244–254. Springer, 2009.

[10] Fook Chin, Qiang Fang, Tao Zhang, and Irena Cosic. A fast critical arrhythmic ecg waveform identification method using cross-correlation and multiple template matching. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, pages 1922–1925, Aug 2010.

[11] Edgar Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[12] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph Hellerstein, and Caleb Welton. Mad skills: New analysis practices for big data. *Proc. VLDB Endow.*, 2(2):1481–1492, August 2009.

[13] Oswaldo Cuen-Téllez and Inés Vega-López. Towards a formal model for ecg data analysis and decision making. In *Fourth International Workshop on Knowledge Discovery, Knowledge Management and Decision Support*. Atlantis Press, 2013.

[14] Michele Dallachiesa, Themis Palpanas, and Ihab Ilyas. Top-k nearest neighbor search in uncertain data series. *Proc. VLDB Endow.*, 8(1):13–24, September 2014.

[15] Ronald Fagin, Ravi Kumar, and Dandapani Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 301–312, New York, NY, USA, 2003. ACM.

[16] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. *Fast subsequence matching in time-series databases*, volume 23. ACM, New York, NY, USA, May 1994.

[17] Association for the Advancement of Medical Instrumentation. Cardiac monitors, heart rate meters, and alarms [american national standard (ansi/aami ec13:2002)], 2002.

[18] Adam Gacek and Witold Pedrycz. *ECG signal processing, classification and interpretation: a comprehensive framework of computational intelligence*. Springer Publishing Company, Incorporated, 2014.

[19] Héctor Garcia-Molina, Jeffrey Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[20] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[21] Ary Goldberger, Luis Amaral, Leon Glass, Jeffrey Hausdorff, Plamen Ivanov, Roger Mark, Joseph Mietus, George Moody, Chung-Kang Peng, and Eugene Stanley. Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.

[22] Scott Greenwald. *The Development and Analysis of a Ventricular Fibrillation Detector*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1986.

[23] Scott Greenwald. *Improved Detection and Classification of Arrhythmias in Noise-corrupted Electrocardiograms Using Contextual Information*. Harvard University–MIT Division of Health Sciences and Technology, Program in Medical Engineering and Medical Physics, 1990.

[24] Alan Halverson, Jennifer Beckmann, Jeffrey Naughton, and David Dewitt. A comparison of c-store and row-store in a common framework. Technical report, University of Wisconsin-Madison, 2006.

[25] John Hampton. *The E. C. G. made easy [by] John R. Hampton*. Churchill Livingstone, [Represented by Longman, New York] Edinburgh, 1973.

[26] M.R. Homaeinezhad, S.A. Atyabi, E. Tavakkoli, H.N. Toosi, A. Ghaffari, and R. Ebrahimpour. Ecg arrhythmia recognition via a neuro-svm–knn hybrid classifier with virtual qrs image-based geometrical features. *Expert Systems with Applications*, 39(2):2047 – 2058, 2012.

[27] Michael Jaedicke and Bernhard Mitschang. User-defined table operators: Enhancing extensibility for ordbms. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 494–505, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[28] Shrikant Kashyap and Panagiotis Karras. Scalable knn search on vertically stored time series. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1334–1342, New York, NY, USA, 2011. ACM.

[29] Manpreet Kaur and Birmohan Singh. Comparison of different approaches for removal of baseline wander from ecg signal. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ICWET '11, pages 1290–1294, New York, NY, USA, 2011. ACM.

[30] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296. IEEE, 2001.

[31] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series: A survey and novel approach. *Data mining in time series databases*, 57:1–22, 2004.

[32] Setrag Khoshafian, George Copeland, Thomas Jagodis, Haran Boral, and Patrick Valduriez. A query processing strategy for the decomposed storage model. In *Proceedings of the Third International Conference on Data Engineering*, pages 636–643, Washington, DC, USA, 1987. IEEE Computer Society.

[33] Vessela Krasteva and Irena Jekova. Qrs template matching for recognition of ventricular ectopic beats. *Annals of Biomedical Engineering*, 35(12):2065–2076, 2007.

[34] Pablo Laguna, Roger Mark, Ary Goldberg, and George Moody. A database for evaluation of algorithms for measurement of qt and other waveform intervals in the ecg. In *Computers in Cardiology 1997*, pages 673–676, Sep 1997.

[35] Dustin Lange and Felix Naumann. Efficient similarity search: arbitrary similarity measures, arbitrary composition. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1679–1688, New York, NY, USA, 2011. ACM.

[36] Sandra Lara-Dévora. Desarrollo de algoritmos para el reconocimiento de patrones de interés en electrocardiogramas. 2012.

[37] Dan Li, Admir Djulovic, and Jianfeng Xu. A study of knn using icu multivariate time series data. In *Proceedings of the International Conference on Data Mining (DMIN)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[38] Jessica Lin, Eamonn Keogh, Ada Fu, and Helga Van. Approximations to magic: finding unusual medical time series. In *Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems*, CBMS '05, pages 329–334, Washington, DC, USA, 2005. IEEE Computer Society.

[39] Daniel López-Barrón, Inés Vega-López, and Oswaldo Cuen-Téllez. Supporting query by content on ecg data with user defined functions. In *Advances in Computer Science (ACS 2013)*, pages 392 –399, April 2013.

[40] Jim Melton and Alan Simon. *SQL:1999: Understanding Relational Language Components*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[41] George Moody and Roger Mark. The impact of the mit-bih arrhythmia database. *Engineering in Medicine and Biology Magazine, IEEE*, 20(3):45–50, 2001.

[42] George Moody, Roger Mark, and Ary Goldberger. Physionet: a web-based resource for the study of physiologic signals. *IEEE Eng Med Biol Mag*, 20(3):70–75, 2001.

[43] George Moody, Roger Mark, and Ary Goldberger. Physionet: Physiologic signals, time series and related open source software for basic, clinical, and applied research. In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, pages 8327–8330. IEEE, 2011.

[44] Yang Moon, Kyu Whang, and Wook Han. General match: a subsequence matching method in time-series databases based on generalized windows. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 382–393, New York, NY, USA, 2002. ACM.

[45] Yang Moon, Kyu Whang, and Woong Loh. Duality-based subsequence matching in time-series databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 263–272, Washington, DC, USA, 2001. IEEE Computer Society.

[46] Carlos Ordonez. Integrating k-means clustering with a relational dbms using sql. *IEEE Trans. on Knowledge and Data Engineering*, 18(2):188–201, February 2006.

[47] Carlos Ordonez. Building statistical models and scoring with udfs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1005–1016. ACM, 2007.

[48] Carlos Ordonez. Statistical model computation with udfs. *Knowledge and Data Engineering, IEEE Transactions on*, 22(12):1752–1765, 2010.

[49] Carlos Ordonez and Javier García-García. Vector and matrix operations programmed with udfs in a relational dbms. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 503–512, New York, NY, USA, 2006. ACM.

[50] Carlos Ordonez and Sasi Pitchaimalai. One-pass data mining algorithms in a dbms with udfs. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1217–1220, New York, NY, USA, 2011. ACM.

[51] Al Qawasmi, Abdel Rahman, and Khaled Daqrouq. Ecg signal enhancement using wavelet transform. *WSEAS Transactions on Biology and Biomedicine*, (2):62–71, 2010.

[52] Hari Rai, Anurag Trivedi, and Shailja Shukla. Ecg signal processing for abnormalities detection using multi-resolution wavelet transform and artificial neural network classifier. *Measurement*, 46(9):3238–3246, 2013.

[53] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, Inc., New York, NY, USA, 3 edition, 2003.

[54] Sweta Saxena, Adrija Sharma, and Subash Chaudhary. Data compression and feature extraction of ecg signals. *International Journal of Systems Science*, 28(5):483–498, 1997.

[55] Kasar Smita and Joshi Madhuri. Article: Ecg signal processing: A survey. *IJCA Proceedings on International Conference in Computational Intelligence (ICCIA 2012)*, ICCIA(7):–, March 2012. Full text available.

[56] Leif Sörnmo and Pablo Laguna. *Bioelectrical signal processing in cardiac and neurological applications*. Academic Press, 2005.

[57] Mike Stonebraker, Daniel Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amersin Lin, Samuel Madden, et al. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.

[58] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. Srs: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proc. VLDB Endow.*, 8(1):1–12, September 2014.

[59] Leif Sörnmo and Pablo Laguna. *Electrocardiogram (ECG) Signal Processing*. John Wiley & Sons, Inc., 2006.

[60] Alessandro Taddei, Alexander Distante, Michele Emdin, Paola Pisani, George Moody, Cees Zeelenberg, and Carlo Marchesi. The european st-t database: standard for evaluating systems for the analysis of st-t changes in ambulatory electrocardiography. *European Heart Journal*, 13(9):1164–1172, 1992.

[61] Pawel Tadejko and Waldemar Rakowski. Mathematical morphology based ecg feature extraction for the purpose of heartbeat classification. In *Computer Information Systems and Industrial Management Applications, 2007. CISIM'07. 6th International Conference on*, pages 322–327. IEEE, 2007.

[62] Xiao Tang and Lan Shu. Classification of electrocardiogram signals with rs and quantum neural networks. *International Journal of Multimedia and Ubiquitous Engineering*, 9(2):363–372, 2014.

[63] Inés Vega-López and Bongki Moon. A quantization approach for efficient similarity search on time series data. In *Proceedings of the International Conference on Internet Information Retrieval*, pages 182–189. Citeseer, 2004.

[64] Inés Vega-López and Bongki Moon. Quantizing time series for efficient subsequence matching. In *Proceedings of the 24th IASTED International Conference on Database and Applications*, DBA'06, pages 209–214, Anaheim, CA, USA, 2006. ACTA Press.

[65] Janis Voigtländer and Armin Kühnemann. Composition of functions with accumulating parameters. *J. Funct. Program.*, 14(3):317–363, May 2004.

[66] Xiaoli Wang, Xiaofeng Ding, Anthony Tung, and Zhenjie Zhang. Efficient and effective knn sequence search with approximate n-grams. *Proceedings of the VLDB Endowment*, 7(1):1–12, 2013.

[67] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ratanamahatana. Fast time series classification using numerosity reduction. In *Procee-*

dings of the 23rd international conference on Machine learning, pages 1033–1040. ACM, 2006.

[68] Kiyoung Yang and Cyrus Shahabi. An efficient k nearest neighbor search for multivariate time series. *Inf. Comput.*, 205(1):65–98, January 2007.

[69] Manisha Zade, Priyanka Palkar, Prafulla Aerkewar, and Agha Pathan. Detection of ecg signal: A survey. *IJAIM*, 1(5):126–130, 2013.

[70] Qibin Zhao and Liqing Zhang. Ecg feature extraction and classification using wavelet transform and support vector machines. In *Neural Networks and Brain, 2005. ICNN&B'05. International Conference on*, volume 2, pages 1089–1092. IEEE, 2005.